UNIVERSITY OF CALIFORNIA
SANTA CRUZ

ROUTING STRATEGIES IN AD-HOC WIRELESS NETWORKS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Jyoti Raju

December 2002

The Dissertation of Jyoti Raju
is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Patrick Mantey

Professor Katia Obraczka

Frank Talamantes
Vice Provost and Dean of Graduate Studies

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
DEC 2002		**00-12-2002 to 00-12-2002**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Routing Strategies in Ad-Hoc Wireless Networks	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**171**	

Contents

List of Figures

List of Tables

Abstract

Routing Strategies in Ad-Hoc Wireless Networks

by

Jyoti Raju

Ad-hoc wireless networks present a unique design problem for routing. Wireless networks suffer from low bandwidth due to high rates of interference and inherent limitations of the medium. Mobility also increases the bandwidth used for control packets. To conserve on precious bandwidth, routing protocols should generate as few updates as possible. In this dissertation, we propose distance vector solutions to ad-hoc routing because unlike existing routing solutions our solutions do not use sequence numbers and thus are not prone to inefficient or wrong behavior in the presence of node failures.

First, we introduce ROAM, the first protocol to correctly tackle the "searching to infinity" problem found in on-demand routing protocols. ROAM can be used in networks with low rates of topology changes because it required reliable updates.

Next, we describe two protocols DST (on-demand) and BEST (table-driven) for networks with high rates of topology changes. Simulation experiments carried out in two different simulation packages show that these protocols perform an order of magnitude better than representative on-demand and table-driven routing solutions for ad-hoc networks.

Finally, we introduce MDST, an on-demand protocol that extends the source tracing algorithm used in DST to create and maintain multiple paths in an ad-hoc

wireless network. Multipath routing can be used in ad hoc networks to achieve greater resilience to route failures and better end-to-end delays. Multipath routing is also essential when using QoS metrics like delay in order to avoid route oscillation. We show that multiple paths that are node disjoint and loop free can be maintained with less overhead than DST. Further, these multiple paths decrease the delay of packet delivery and increase the throughput of the network.

To Avatar Meher Baba and Sadguru Sri Meher Chaitanyaji Maharaj

Acknowledgements

I am very grateful to my advisor, Prof. J.J. Garcia-Luna-Aceves for his invaluable guidance and support. His sense of humour and patience made working with him a joyous experience. Thank you JJ, for giving me the inspiration and the opportunity.

I would like to thank Prof. Pat Mantey and Prof. Katia Obraczka for being part of my committee and for their valuable feedback. Thanks also to Prof. Martine Schlag and Prof. Anujan Varma for teaching excellent classes.

I will always cherish the years in UCSC and the friends I made here. Thanks to Carol Mullane for all her help and friendly chats and to Julie Kimball for the great yoga and swimming classes, which were the only things that got me out of bed some mornings. My gratitude to Buzz and Ginger for "adopting" me and opening up their wonderful life and home to me. To Chris, Abigail, Shree, Pratibha, Sharon, Lori, Suzana, Mike, Ewerton, Daniela, Jose, Soumya, Brad, Marcelo, Lichun, Chane, Heather, Emily, a big thank you, for making UCSC so enjoyable. Thanks to all of RAF, especially the Bay Area Gang, Deepa, Jyotsna and Lavanya for eleven unbroken years of fun. Life would be very boring without the innumerable chai breaks provided by Sangeeta, Anil, Rohitash, Rashmi, Rajesh, Amita and lil' Ishani and Varun. Thank you guys!

This thesis would not have been possible without the love, affection and nagging of my beautiful mother, my father, my grandfather, my best friend and sister, Swati, and my dearest husband, Varma. To them go my deepest thanks.

This thesis is the result of the divine blessings of Avatar Meher Baba and

Sadguru Sri Meher Chaitanyaji Maharaj and is dedicated to them.

The text of this dissertation includes material that has been previously published in [41, 43, 42, 45] and [44]. Professor J.J. Garcia-Luna-Aceves, listed as the co-author in these publications, directed and supervised the research that forms the basis for this dissertation.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant DAAB07-95-C-D157.

Chapter 1

Introduction

As access to the Internet becomes more and more widespread, many technologies besides the traditional phone lines are being used as conduits to the Internet. The wireless option is particularly attractive as it has the most potential to make "anytime, anywhere" access feasible.

The architecture of wireless networks has proceeded in two main directions. The first model is based on traditional cellular networks where all nodes are one-hop away from a base station, which is usually attached to the Internet with a wired connection. As shown in Fig. 1.1.a, all nodes use the base station to make connections to and from the Internet. There can only be peer-to-peer connection through the base station. As the mobile nodes move out of range of one base station and into the range of another, a "handoff" occurs from the old base station to the new, and the mobile node is able to communicate seamlessly with the network. Since a mobile node is always one hop away from the base station, routing is not needed in this

architecture. Typical applications of this type of network include office automation using wireless local area networks (WLANs), examples of which are the recent WAP products introduced by commercial vendors like Apple and Nokia.

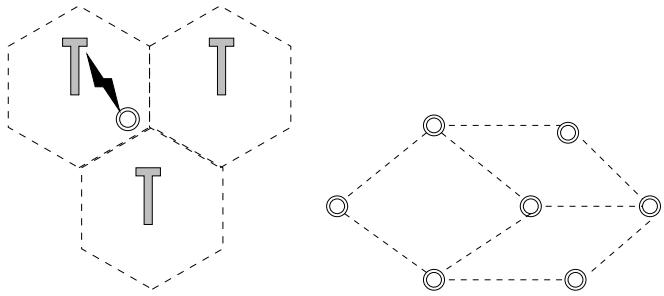


Figure 1.1: Examples of the two models

The second model is shown in Fig. 1.1.b and is used when there is no wired infrastructure in place. Typical applications of this model include disaster relief, military scenarios and networks set up at temporary events like a class lecture or business convention. Such networks are termed as *ad-hoc wireless networks*. All ad-hoc networks share the following defining features:

- Not all nodes are within line of sight of each other or a base station. Thus, packets may have to be relayed several times over the multiple-access channels.
- Nodes can serve as sources, relays and destinations of data traffic.
- Due to limited transmission range, mobility causes frequent changes in connectivity, i.e., the network topology is dynamic.

Due to the multihop and dynamic nature of ad-hoc networks, a distributed routing protocol is required to forward packets between nodes, and to and from the

Internet.

Nodes in an ad-hoc network can easily run routing protocols designed for wired networks, provided the nodes contain proper protocol stacks. However, specific features like low bandwidth and high rates of interference present a design problem which is very different from routing in wired networks. A large body of research exists to tackle these specific issues.

1.1 Routing in wireless ad-hoc networks

Work on ad-hoc network routing started as early as the 70's with research on the DARPA PRNET project [14, 46, 30]. Since then, numerous protocols have been developed that take into account the characteristics of wireless networks. Routing for wireless ad-hoc networks can generally be categorized as either table-driven (*proactive*) or on-demand (*reactive*) routing.

Table driven routing protocols attempt to maintain consistent up-to-date routing information about the distance to every other node in the network. These protocols react proactively by sending update messages in response to topology changes. Destination Sequenced Distance Vector Protocol (DSDV) [39] and Wireless Routing Protocol (WRP) [35] are two widely known table driven routing protocols for wireless networks. DSDV is based on the classic Bellman-Ford routing mechanism [28], in addition to which it uses sequence numbers to distinguish stale routes from new routes, thus avoiding the formation of loops. WRP, on the other hand avoids the "counting-to-infinity" problem by maintaining second-to-last hop information about

every destination node. However, as shown in [7], table driven routing protocols that aim to maintain consistent information at all times usually suffer from high control overhead.

The IETF working group for Mobile, Ad hoc Networking (MANET) has provided impetus for routing protocols in ad-hoc networks. Most of the routing protocols proposed in the MANET working group are on-demand routing protocols. On-demand routing protocols were designed with the aim of reducing control overhead, thus increasing bandwidth and conserving power at the mobile stations. These protocols limit the amount of bandwidth consumed by maintaining routes to only those destinations for which a source has data traffic. Therefore, the routing is source-initiated as opposed to table-driven routing protocols that are destination initiated. Most on-demand routing protocols do non-optimum routing, i.e., a route does not necessarily have to be the shortest path from a source to a destination in order to be used for forwarding packets; a route is used as long as it is valid. This feature also reduces control overhead.

There are several recent examples of the on-demand routing approach. These examples include Ad-Hoc On-Demand Distance Vector (AODV)[40], Associativity Based Routing (ABR)[49], Dynamic Source Routing (DSR) [27], Temporally Ordered Routing Algorithm (TORA) [38] and Zone Routing Protocol (ZRP) [24]. The routing protocols differ on the specific mechanisms used to disseminate flood-search packets and their responses, cache the information heard from other nodes' searches, determine the cost of a link, and determine the existence of a neighbor. However, all the

on-demand routing proposals use flood search messages that either: (a) give sources the entire paths to destinations, which are then used in source-routed data packets (e.g., DSR); or (b) provide only the distances and next hops to destinations, validating them with sequence numbers (e.g., AODV) or time stamps (e.g., TORA).

1.2 Issues specific to wireless ad-hoc networks

Five key issues need to be addressed when designing a routing solution for wireless ad-hoc networks. First, changes in topology may cause a routing protocol to create permanent loops in its tables. A classic example of this is the permanent looping caused in Routing Information Protocol (RIP) [25]. When a loop is formed, data packets keep traversing the loop, causing congestion and using up resources like buffer space and link bandwidth. A wireless network running an algorithm that does not prevent permanent loops is even more susceptible because of the high rate of topology change. A by product of congestion is the reduction of battery life of routers, causing partition of the network in extreme cases.

Second, with most of today's on-demand routing protocols, when a destination fails or becomes unreachable from a network component, a source trying to obtain a path to the destination finds that its flood-search for the destination fails, but is unable to determine whether or not it should start another flood search. The search could have failed simply due to temporary link failures induced by fading or node mobility, for example. There are no inherent mechanisms in these on-demand routing protocols that would prevent a source from repeating its search in the event

that the destination is not reachable, which we call the *searching-to-infinity* problem. This problem also makes the (wired or wireless) network running an on-demand routing protocol susceptible to a unique form of attack, where a malicious router can indefinitely query a network for a destination that does not exist, thus causing congestion due to queries. Consequently, external mechanisms are used today in order to stop sources from sending unnecessary queries. In DSR and AODV, routers do not keep state about the search queries in progress, and the application accessing the on-demand routing service must implement a hold-down time after a search fails; however, just as is the case in Cisco's IGRP [25], it is difficult to determine an adequate length of hold-down time or how many times a source should persist requesting a path to a destination. In addition, each source must go through the process independently.

Third, solutions that depend on sequence numbers to maintain loop-freedom (AODV) or prevent searching to infinity (TORA) are usually not fault-tolerant. When a node fails and restarts, it resets its sequence number and this will lead to a situation where its updates are discarded until the sequence number catches up to the highest advertised value. Given that there will be higher rates of node failures in ad-hoc networks, it is imperative that a routing solution should not stop working when nodes fail.

Fourth, wireless networks suffer from low bandwidth and high rates of interference. To conserve on precious bandwidth, routing protocols should generate as few updates as possible. Mobility also increases the bandwidth used for control packets. As links go up and down frequently, more updates need to be sent to maintain correct

topology information. When congestion due to control overhead increases, the convergence time of the routing algorithm increases, causing more and more data packets to travel along wrong paths and waste more bandwidth. Therefore, bandwidth efficiency is very important in wireless networks. Higher bandwidth efficiency results in higher battery life.

Lastly, as the number of multimedia applications increases, quality of service metrics like delay and jitter become important issues in wireless networks. Multipath routing has been shown as a favorable alternative both for circuit switched and packet switched networks [5, 15, 37, 8, 52, 51, 6, 11] as it provides an easy mechanism to distribute traffic, balance network load as well as increase probability of delivery. These protocols use table-driven routing algorithms (link state or distance vector) to compute multiple routes. On-demand routing protocols lend themselves very naturally to multipath routing because the route discovery process can be tuned to return multiple paths and hence are key to QoS routing in ad-hoc wireless networks.

1.3 Organization of Thesis

Our aim in this thesis is to design protocols that specifically tackle issues encountered in wireless networks. The first design issue we tackled was the "searching to infinity" problem. We designed the Routing On-Demand Acyclic Multipath Protocol (ROAM), which is the only on-demand routing protocol that tackles the searching to infinity problem and has correct behavior in the presence of network partitions. ROAM uses only distances and internodal coordination to maintain loop-freedom at

all times. ROAM does not rely on sequence numbers and hence is inherently fault-tolerant. We also provide the first correctness proof for an on-demand routing protocol that does not allow searching to infinity. Our work on ROAM is presented in Chapter 2.

In the ROAM protocol, we assumed that reliable updates are available to the routing protocol. Reliability can be added at the link layer. However, most existing wireless stack implementations do not have reliable link layers. This prompted us to design Dynamic Source Tracing (DST), which also does not use sequence numbers to prevent permanent looping; it uses information about the second-to-last hop to the destination along with the distance to the destination to prevent permanent loops. DST may have short term loops, but considerable performance gains are attained compared to the internodal coordination approach used in ROAM. We also use the central paradigm of on-demand routing, which is not to send updates as long as there exists at least one path to the destination, even though the path is longer than the shortest possible path. Using simulations, we show that DST is very bandwidth efficient. Since data packets do not have source routes in headers, we will have a considerable increase in bandwidth efficiency compared to protocols like DSR that use them. We also present a proof of correctness for DST. Chapter 3 presents our work on DST.

The excellent results we obtained with DST prompted us to see if a table-driven routing protocol could be adapted to provide the low overhead we see in DST. This table driven routing protocol would not send updates as long as a path to the destination exists, even though the path is not optimal. BEST is our solution and

it is a table driven routing protocol that avoids permanent loops and yet creates the least packet overhead possible. A proof of correctness under the assumption of reliable delivery of updates is also presented. We did a simulation study of various scenarios comparing DST, BEST and DSR. In this study we introduced a scenario where an ad-hoc network is used as an extension to the Internet. The aim was to ascertain if the efficiency of the routing protocol was scenario-driven. BEST, along with the simulations and proof of correctness is presented in Chapter 4.

Lastly, we propose MDST, a multipath on-demand routing protocol that does not rely on sequence numbers or timestamps to achieve correctness. Further, this protocol uses node disjoint paths to achieve better throughput. This protocol and a simulation study comparing it with DST are presented in Chapter 5.

Finally, we present a summary of our work together with conclusions and future research directions in Chapter 6.

Chapter 2

On demand Routing using Diffusing Computations

Prior work in on-demand routing has followed three main approaches to ensuring that the routes obtained are free of long-term loops. Loops are formed when a router picks an upstream neighbor from it as the next hop towards a destination.

The Dynamic Source Routing (DSR) protocol [27] is an example of using complete path information to avoid loops. In DSR, the network is flooded with queries when a source requests a search for a route to a destination. Finding a route results in a reply being sent back in a route reply packet and the resultant routes are stored in a route cache at the sender. The entire route is put in the header of a data packet and this route is used to forward packets.

The ad-hoc on-demand distance vector routing algorithm (AODV)[40] is an example of using sequence numbers to avoid long-term loops. In AODV, each destina-

tion maintains a sequence number that it updates every time there is a connectivity change with its neighbors. A router accepts those routes for a destination that are associated with the largest sequence number received for that destination. In turn, the destination always uses odd sequence numbers, and routers whose routes to the destination break increase the sequence number for the destination and report an infinite distance to it. AODV uses a query flooding technique similar to DSR.

The Temporally-Ordered Routing Algorithm (TORA) [38] is an example of using time stamps and internodal coordination to avoid looping. TORA uses a link-reversal algorithm [19] to maintain loop-free multipaths that are created by a query-reply process similar to the above two algorithms. TORA relies on synchronized clocks to create timestamps that maintain the relative ordering of events.

Both AODV and DSR suffer from the "searching to infinity" problem which is described in Chapter 1. TORA provides a solution for searching to infinity, but TORA uses logical synchronization which would not perform correctly in the presence of partitions unless clocks are sychronized across the network. This is a difficult requirement to fulfill in an ad-hoc network.

The work presented in this section introduces a new approach to the establishment and maintenance of loop-free routes on demand in either wireless networks or wired networks. We present the ROAM (Routing On-demand Acyclic Multipath) algorithm[41], which uses internodal coordination along directed acyclic subgraphs defined solely on the routers' distances to destinations. We call the operations used to coordinate nodes "diffusing computations". ROAM extends the diffusing update

algorithm (DUAL) [20] to provide routing on demand.

In ROAM, a search query in a connected component results in either the source requesting a route to a destination obtaining its answer, or all the routers determining that the destination is unreachable. Hence, ROAM eliminates the need for application-level mechanisms to prevent excessive flooding of searches in the event destinations are not reachable.

2.1 Network Model and Notation used in ROAM

To describe ROAM, we model a network as an undirected graph $G(V, E)$. V is the set of routers in the network and E is the set of links in the network. Each router has a unique ID and a link is said to exist between two routers if they can exchange packets. Each link has two costs associated with it, one in either direction.

Our description and verification of ROAM assumes the existence of a link level protocol which ensures that:

- ROAM is notified about the existence of a new neighbor or the loss of connectivity with a neighbor within a finite time.
- Link costs are always positive and a failed link has infinite cost.
- All control packets are sent reliably and are received within a finite amount of time. If the packets cannot be sent after a specified amount of retries, then the link layer marks the neighbor as being down and sends an indication to the routing protocol. Since control packets travel only one-hop, we only require

single hop reliability.

- All messages and changes in the cost of links, and the addition and deletion of neighbors are processed within a finite time.

- Messages can be transmitted over a link only when the link is perceived as being up.

Reliable message transmission can be easily added into a routing protocol for a wired network [34]. In a wireless network, the logical link control [2] necessary to satisfy the above assumptions can be implemented on top of any MAC protocol designed for wireless links based on collision avoidance (e.g., IEEE802.11), TDMA, or any of the various dynamic scheduling MAC protocols proposed recently [47, 29, 53], without requiring additional network-level control packets.

The following notation is used throughout the description of ROAM:

N: the set of destinations a router knows about.

N_i: the set of routers connected through a link with router i, i.e., the set of neighbors of router i.

l_k^i: the cost of the link to neighbor k; the cost of a failed link is assumed to be ∞.

$D_j^i(t)$: the current distance maintained by router i for destination j at time t.

$D_{jk}^i(t)$: the distance from neighbor k to router j as known by router i at time t.

$FD_j^i(t)$: the *feasible distance* at router i for destination j; this distance is used to check if the *feasibility condition* (defined in Section 2.3) is satisfied.

$RD_j^i(t)$: the distance to destination j used in messages sent to the neighbors at time

t.

$D_j^{*i}(t)$: the smallest value assigned to D_j^i from the time i became passive up to time t.

SS_j^i: is the set of neighbors of router i that offer loop-free routes to destination j; any neighbor k whose distance as known by i, D_{jk}^i is less than the feasible distance FD_j^i belongs to this set.

s_j^i: the *successor* for destination j; this successor offers a loop-free path to destination j and is used for data packets.

o_j^i: the *query origin flag* records how a router gets into the active state (further explanation in section 2.5).

T_j^i: this timestamp is maintained for each destination. It indicates the last time a data packet was seen for the destination.

$ST_{jk}^i(t)$: this value can be set to active or passive; when set to active, it indicates that router i has sent a query to neighbor k and expects it to return a reply for destination j.

2.2 Information stored and exchanged by routers in ROAM

Each router maintains a *distance table*, a *routing table* and a *link-cost table*. The distance table at router i is a matrix containing for each destination j and for each neighbor k of router i, the distance D_{jk}^i as last reported by k and a reply status flag ST_{jk}^i. ST_{jk}^i is set to active if router i has sent a query to router k for destination j but has not received a reply and set to passive otherwise.

The routing table at router i is a column vector containing, for each destination j, the distance to the destination D_j^i, the feasible distance FD_j^i, the reported distance RD_j^i, the successor s_j^i, the query origin flag o_j^i and the timestamp T_j^i. Therefore, if there is no data traffic received for a destination, the destination entry is eventually timed out and removed from the routing table.

The link-cost table lists the costs of links to each known adjacent neighbor (l_k^i). The cost of a link from i to k is denoted as l_k^i and is considered to be infinity when a link fails.

There are three types of control packets used by the routing protocol: queries, replies and updates. A control packet from router i to router k contains the addresses i and k and the address of the destination j for which a path is desired. The packet also contains a field indicating the reported distance (RD_j^i) from router i to destination j. A flag u_j^i indicates whether a control packet is an update, a query or a reply to a query. The distance in a packet can be set to any positive value including infinity.

2.3 Active and Passive States in ROAM

A router i updates its routing table for a destination j when: (a) it needs to add an entry for j, (b) it needs to modify its distance to j (including setting that distance to ∞), and (c) it decides to erase the entry for j.

For a given destination, a router that has sent queries to all its neighbors and is waiting for replies from at least one of its neighbors is said to be active; otherwise, it is said to be passive. With respect to a given destination j, a router running ROAM

can be in one of the following three states: (a) passive knowing or not knowing about j's existence, (b) active waiting to obtain distance information about j (while creating routes), and (c) active waiting for replies from neighbors about a known destination j (while maintaining routes). A router i initializes itself in the passive state with a distance of zero to itself ($D_i^i = FD_i^i = RD_i^i = 0, s_i^i = i, T_i^i$ =present time). A router becomes active or passive for a given destination depending on whether or not at least one of its neighbors has reported a distance to the destination that is short enough to be trusted not to lead to a loop.

To maintain loop-free routes, each router can only pick as successor a neighbor that satisfies either of the two feasibility conditions. To remain passive and have a loop-free route, a router needs to have a neighbor that is a *feasible successor* (fs_j^i). The feasible successor provides the shortest loop-free path to the destination. The passive feasibility condition (PFC) is to be satisfied by a router's successor when a router is passive. The active feasibility condition (AFC) comes into play only when a router is active, i.e., there is no longer any feasible successor. All neighbors in SS_j^i satisfy AFC, which implies that they provide loop-free paths. However, when a router is active, SS_j^i no longer contains the feasible successor.

PFC: If at time t router i needs to change its successor, it can choose as its new successor any neighbor $q \in N_i(t)$ for which $D_{jq}^i(t) + l_q^i(t) = Min\{D_{jx}^i(t) + l_x^i(t)\ \forall x \in N_i(t)\}$ and $D_{jq}^i(t) < FD_j^i(t)$, where $FD_j^i(t) = D_j^{*i}(t)$.

AFC: If at time t router i becomes active, then it can set its successor to any neighbor $q \in N_i(t)$ where $D_{jq}^i(t) < FD_j^i(t)$. If there is no such router, then the router maintains

its earlier successor until it becomes passive again.

The feasible successor plays a key role in maintaining loop-freedom, because it creates a total ordering of distances along any path [20]. Only the distance through the feasible successor is reported in control messages. Therefore, we are able to maintain multiple routes while introducing no extra latency or control messages. Neighbor routers that satisfy AFC and not PFC can be used for forwarding packets even while the router is active or passive, but their distances are not used in path calculations.

Consider Fig. 2.1 in which router j is the destination and routers a, b and c are neighbors of router i. Router b satisfies PFC and therefore is the successor and feasible successor of router i; router a is in the successor set SS_j^i as it satisfies AFC. If link (i, b) fails, router a is marked as successor, even though router c offers a shorter path. This is done because we know that only router a guarantees a loop-free path. However, because the path through a is not the shortest possible, router i becomes active and start a diffusing computation.

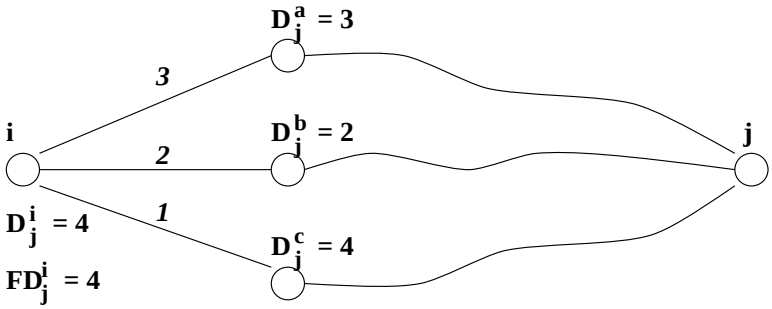


Figure 2.1: Successors in ROAM

As long as a router i finds a successor satisfying PFC after processing an input event, the router does not have to become active; otherwise, router i must start

or forward a diffusing computation and become active. The next few sections describes how diffusing computations are used in ROAM to create, maintain, and delete routes to destinations on demand.

2.4 Creating Routes

When a router gets a data packet for a destination for which it has no entry in its routing table, it starts a *diffusing search*, which is a diffusing computation originated by a source and propagated by each router that has no entry for the destination. The source of this search can be either the source of the data packet or any intermediate router on the path from the source to the destination. The diffusing search propagates from the source out on a hop by hop basis, until it reaches a router that has an entry for the requested destination, in which case the router replies with its distance to it. At the end of the search, the source either obtains a finite distance to the destination or all the nodes in the same connected component determine that the destination is unreachable ($D_j^i = \infty$ and node is passive).

A router starting the diffusing search adds the destination to its routing table ($D_j^i = FD_j^i = RD_j^i = \infty$, $s_j^i = null$, $o_j^i = 1$, $T_j^i =$ present time) and distance table ($D_{jk}^i = \infty \ \forall k \in N_i$), becomes active for the destination ($ST_{jk}^i =$active $\forall k \in N_i$) and sends a query to its neighbors. The queries used in a diffusing search report a distance $RD_j^i = \infty$.

A neighbor i that receives a query for j and has no entry for the destination adds the destination to its routing table ($D_j^i = FD_j^i = RD_j^i = \infty$, $s_j^i = null$, $o_j^i = 3$,

T_j^i = present time) and distance table ($D_{jk}^i = \infty \ \forall k \in N_i$), becomes active for the destination (ST_{jk}^i =active $\forall k \in N_i$) and forwards the query to its neighbors.

Replies to a query can result in making active routers passive and therefore shrinking the diffusing search and finally ending it. When a router gets a reply from neighbor k, it records the reported distance ($D_{jk}^i = RD_j^k$) and resets the active flag (ST_{jk}^i =passive).

Replies are sent by routers when any of the following three conditions are satisfied:

1. A router that already has an entry for the destination, infinite or finite, sends back a reply immediately with $RD_j^i = D_j^i$, because PFC is satisfied already. This condition also holds for the destination of the diffusing computation.

2. A router that is already active for the destination sends a reply back immediately with $RD_j^i = D_j^i$.

3. A router i other than the source of the diffusing search that has received replies from all its neighbors sends a reply with $RD_j^i = D_j^i$. Before sending the reply, i sets ST_{jk}^i =passive for all $k \in N_i$ and sets its feasible distance, reported distance and distance to the minimum value of $D_{jk}^i + l_k^i$ for all $k \in N_i$. The neighbor that offers the minimum value becomes the new successor and feasible successor. If all the reply distance values received by a router are infinity, then the router sends a reply with $RD_j^i = \infty$ to the neighbor that sent it the query.

Fig. 2.2 shows a diffusing computation where router i is searching for a path to router

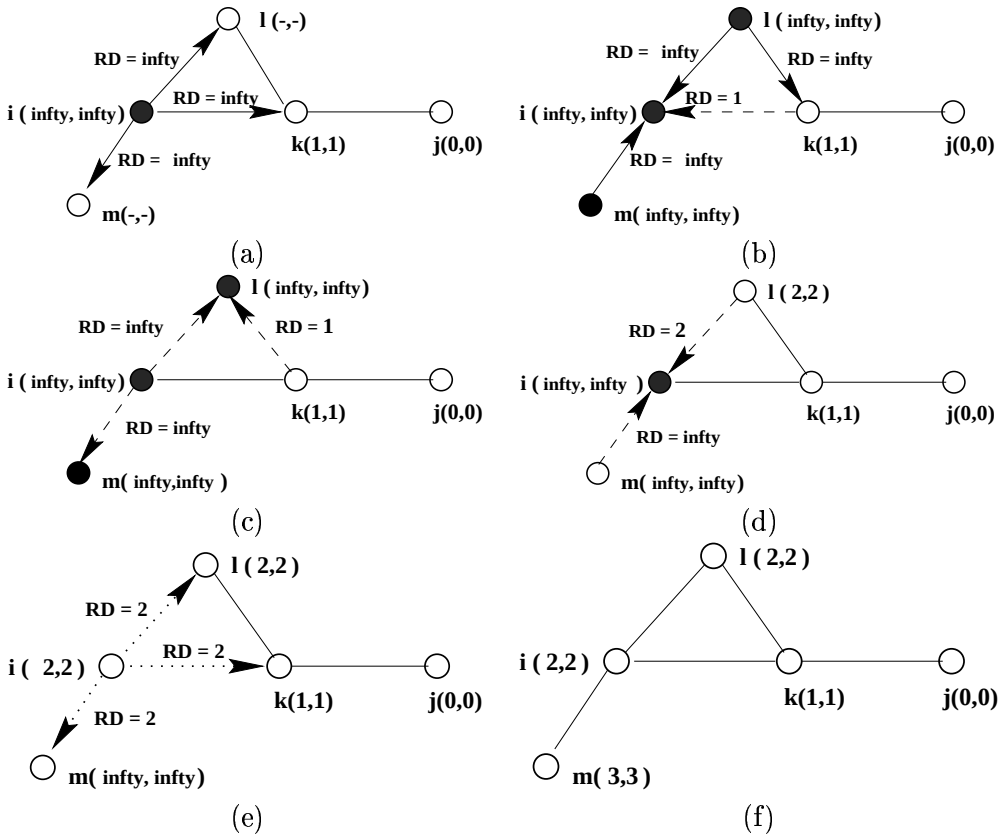


Figure 2.2: Creating routes: Router i searches for destination j

j. For simplicity, all link costs are assumed to be 1. The first entry in the parenthesis is the distance to destination j and the second entry is the feasible distance to destination j. Routers k and j are the only ones who know of router j's existence. The queries are denoted by arrows with solid lines. The arrows with dashed lines are the replies and the dotted arrows are updates. Black circles are routers that are active and white circles are routers that are passive.

If the source router gets a finite distance after a search, there can exist certain areas of the network that did not receive replies confirming the existence of the destination. In Fig. 2.2, router m would correspond to such a router. These routers would assume that they are partitioned from the destination because they still have a distance of infinity to the destination. To avoid this condition, we incorporate a mechanism called *threshold updates*. These updates are sent by a router when its distance to a certain destination changes by more than a defined threshold ΔD. The parts of the network that have infinite entries for a destination that is not partitioned eventually change their distances to the correct distance. Routers that have no entry for the destination do not propagate these updates.

2.5 Handling Link Cost Changes

Link cost changes to a router k that is not the feasible successor just involve updating entries in the link cost table. When a link cost $l^i_{s^i_j}$ to the feasible successor s^i_j decreases, router i just updates the distance and the feasible distance to reflect the new value $FD^i_j = D^i_j = D^i_{js^i_j} + l^i_{s^i_j}$. If $|D^i_j - RD^i_j| > \Delta D$, RD^i_j is set to the new value

of D_j^i and a threshold update is broadcast to all neighbors.

When a link cost to a feasible successor increases, router i checks to see if any other neighbor in SS_j^i still satisfies PFC and therefore can be made the new feasible successor. If PFC is not satisfied, then router i becomes active and starts a diffusing computation for destination j. Before sending out queries, the router checks to see if SS_j^i is non-empty, in which case it picks a neighbor m in SS_j^i as its successor. The reported distance and distance are set to the distance through m. Therefore, the queries contain the distance through m. However, the feasible distance is not changed. If SS_j^i is empty, then the reported distance, feasible distance and distance is set to the new distance through the original successor (the successor that was the feasible successor).

Once router i starts a diffusing computation for destination j, it sets its flags ST_{jk}^i to active and sends queries to all its neighbors. ST_{jk}^i remains active until a reply from k is received. Therefore, if ST_{jk}^i is set to active for any neighbor k, then router i does not forward any further queries, thus ensuring that the queries are not forwarded forever. This mechanism also helps separate different diffusing computations for the same destination.

When an active router gets replies from all its neighbors, it resets FD_j^i to infinity. It then picks the neighbor that satisfies PFC as the new feasible successor and sets its feasible distance, distance and reported distance equal to the distance through the new feasible successor. A router behaves differently if there have been distance increases while it was active, as explained in the following paragraph.

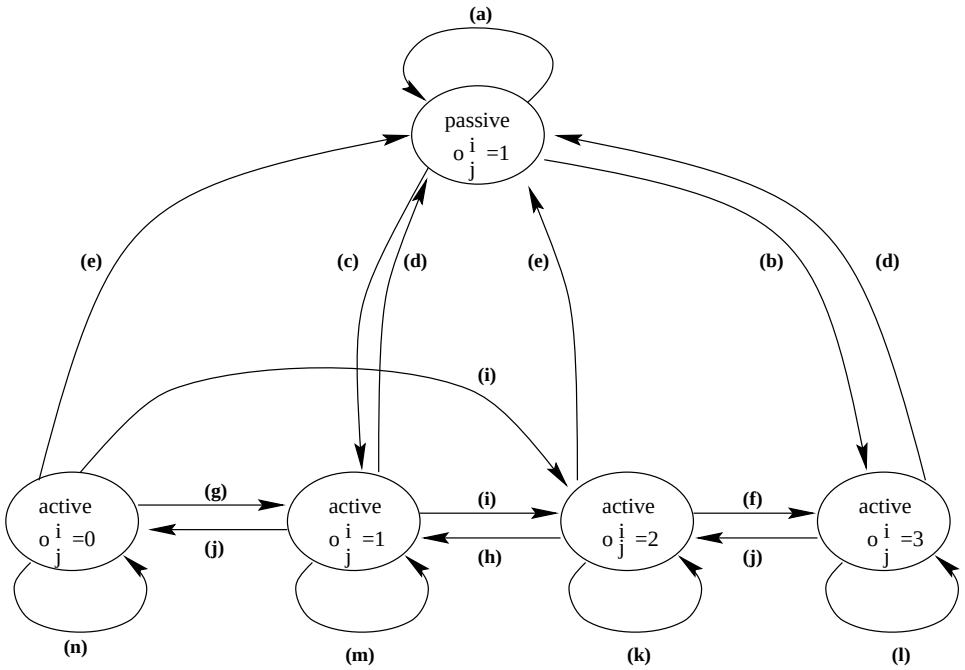


Figure 2.3: Active and Passive states in ROAM

ROAM makes sure that, for any given destination, a router takes part in only one diffusing computation at a time. However, there might be more than one distance increase that needs to be processed while a router is active. To keep track of the multiple inputs a router may have to process, the query origin flag o_j^i is maintained by every router i for every destination j. This flag is set to 1 when a router is passive (ST_{jk}^i = passive $\forall k \in N_i$). When a router is active (ST_{jk}^i = active for some $k \in N_i$) the value of o_j^i can imply the following conditions. It must be noted that a router may get queries from any neighbor, but it becomes active only when the feasible successor no longer satisfies PFC.

- $o_j^i = 0$: Router i is the origin of the query in progress and it has experienced at least one distance increase since becoming active.

(a) delete update or
input event related to nbr k which is not a successor or PFC satisfied or D^i_j = infinity and $D^i_{j\,k}$ = infinity

(b) query from successor and PFC and AFC not satisfied

(c) input event other than query from successor and PFC not satisfied or query from successor with AFC satisfied and PFC not satisfied.

(d) last reply ; *action:* set FD^i_j = infinity

(e) last reply and PFC satisfied with current value of FD^i_j

(f) last reply and PFC and AFC not satisfied with current value of FD^i_j

(g) last reply and PFC not satisfied with current value of FD^i_j

(h) last reply, PFC not satisfied but AFC satisfied.

(i) query from successor

(j) increase in D^i_j

(k) input event other than last reply

(l) input event other than last reply or increase in D^i_j

(m) input event other than last reply, increase in D^i_j or query from successor

(n) input event other than last reply or query from successor.

Figure 2.4: Active and Passive states in ROAM - Legend

- $o_j^i = 1$: Router i is the origin of the query in progress and it has experienced no distance increase and no query from successor since becoming active.

- $o_j^i = 2$: Router i became active due to a query from a successor and it experiences a distance increase, or it is the origin of a query and receives a query from the successor after becoming active.

- $o_j^i = 3$: Router i becomes active after receiving a query from a successor and experiences no distance increases after becoming active.

When router i changes state from active to passive and $o_j^i = 1$ or 3, router i resets the value of FD_j^i to infinity. This results in router i's picking as feasible successor the neighbor that offers the shortest path. If on the other hand $o_j^i = 0$ or 2, router i retains its old FD_j^i and checks for PFC. If PFC is not satisfied, another diffusing computation is started. Before starting the diffusing computation, the values of o_j^i are changed from 0 to 1 and 2 to 3 respectively. Thus, we see that all distance increases are taken care of. A distinction is made between $o_j^i = 1$ and 3 because in the case of $o_j^i = 3$, a reply needs to be sent back to the old successor before the router becomes passive. A parallel distinction can be drawn between $o_j^i = 0$ and 2. Fig. 2.3 shows the states in ROAM and the transitions between them. The figure does not consider link failures and link additions, which are discussed in the next section.

2.6 Handling topology changes

The topology of the network can change by links going down or links coming up. When a new link comes up, it could result in partitioned sections of the network coalescing. Links going down may result in a network getting partitioned besides destroying existing routes. The failure of a router can be viewed as multiple links going down.

If router i detects a new neighbor k, it adds the neighbor to its routing table if the neighbor is a new destination ($D^i_k = FD^i_k = RD^i_k = \infty$, $s^i_k = k$, $o^i_j = 1$, $T^i_k =$ present time). An entry for k is created in the distance table ($D^i_{jk} = \infty \ \forall j \in N$) and a full-state update is sent to the new neighbor. The full-state update packet contains entries for all destinations contained in router i's routing table. If router i is passive for a destination, then the entry is marked as an update, else it is marked as a query; an exception being routing entries with distance infinity which are marked as queries. The reason for this exception, given that routes are set on demand, can be explained using Fig. 2.5.

Consider three networks A, B and C joining. All the routers in A have the distance to destination j set to infinity. The routers in B have no entry for j and the routers in C have a finite entry for j. When the link connecting A and B comes up, if the entry for j is a simple update, then the router in B will ignore it. Therefore, even though there is a route to get to destination j which is in component C, routers in A will never be able to reach it because all of them have their distances set to infinity. Now, if the entry is a query, a diffusing search takes place in component B, at the end

of which routers in A and B know the correct distance to destination j. The full-state

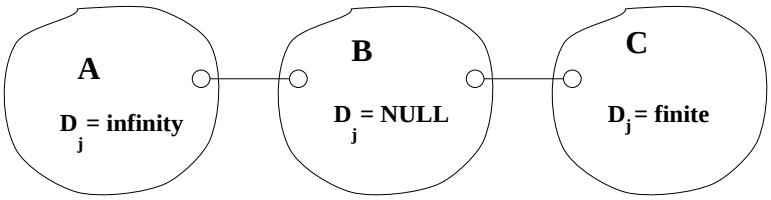


Figure 2.5: Networks with different states coalescing

update can be split into multiple update packets if it does not fit into one. When a router i receives a full-state update packet from a neighbor k, it processes each entry one by one. A query entry is processed in the manner described in sections 2.4 and 2.5.

If the entry is a query about destination j and the router does not already have an entry for that destination, then i adds the destination to its routing table ($D_j^i = FD_j^i = RD_j^i = \infty$, $s_j^i = k$, $o_j^i = 3$, T_j^i =present time) and distance table ($D_{jm}^i = \infty \ \forall m \in N_i, m \neq k$ and $D_{jk}^i = RD_j^k | m = k$), becomes active for the destination (ST_{jk}^i =active $\forall k \in N_i$) and forwards queries to its neighbors. If the query is for a destination that is in router i's tables, the reported distance is stored ($D_{jk}^i = RD_j^k$) and if PFC is not satisfied, router i becomes active (ST_{jk}^i =active $\forall k \in N_i$, $o_j^i = 3$) and sends queries to all its neighbors. If PFC is satisfied, then router k remains passive and sends back a reply to router i containing its distance to j.

If the entry is an update for a destination that i has no knowledge of, then i simply ignores the entry, else it records the distance ($D_{jk}^i = RD_j^k$). If the distance through the neighbor is greater than the present distance ($D_{jk}^i + l_k^i > D_j^i$), nothing is done. If the distance through the neighbor is smaller than the present distance then

router i sets router k as its new feasible successor ($FD_j^i = D_j^i = D_{jk}^i + l_k^i, s_j^i = k$). If the change in distance is greater than a threshold value, router i sends its neighbors the new distance in updates.

If a failure of link (i,k) is detected at router i, router i sets the value D_{jk}^i to infinity for each destination j. If router i was active at the time of deletion of link (i,k), then setting ST_{jk}^i to false and D_{jk}^i to infinity mimics the behavior that would result from router i getting a reply with distance set to infinity from router k. If i was passive and k happened to be the feasible successor, then router i becomes active and starts a diffusing computation. If (i,k) was the only link connecting router i's component and router k's component, then with the loss of link (i,k) router i loses its only successor. This results in router i's sending a query with distance set to infinity. Since this query propagates to all routers in router i's component, all of them eventually change their routing table entries to infinity, which signifies partition from the destination in router k's component.

Fig. 2.6 shows an example of a network where links go down. The topology and notation of the example is same as that in Fig. 2.2 except that the two links (i,k) and (i,l) have link costs of 2. In Fig. 2.6 (a) the link (i,k) fails where k is the feasible successor of i. At router i, the feasible distance is 3. Router l satisfies the feasibility condition since its distance 2 is lesser than 3 and it now offers the shortest path to the destination. Therefore, router i remains passive and changes its distance to 4. Note, however that the feasible distance does not change as it is defined to be the lowest distance value since the router became passive. In Fig. 2.6 (b), link (i,l) fails. Now,

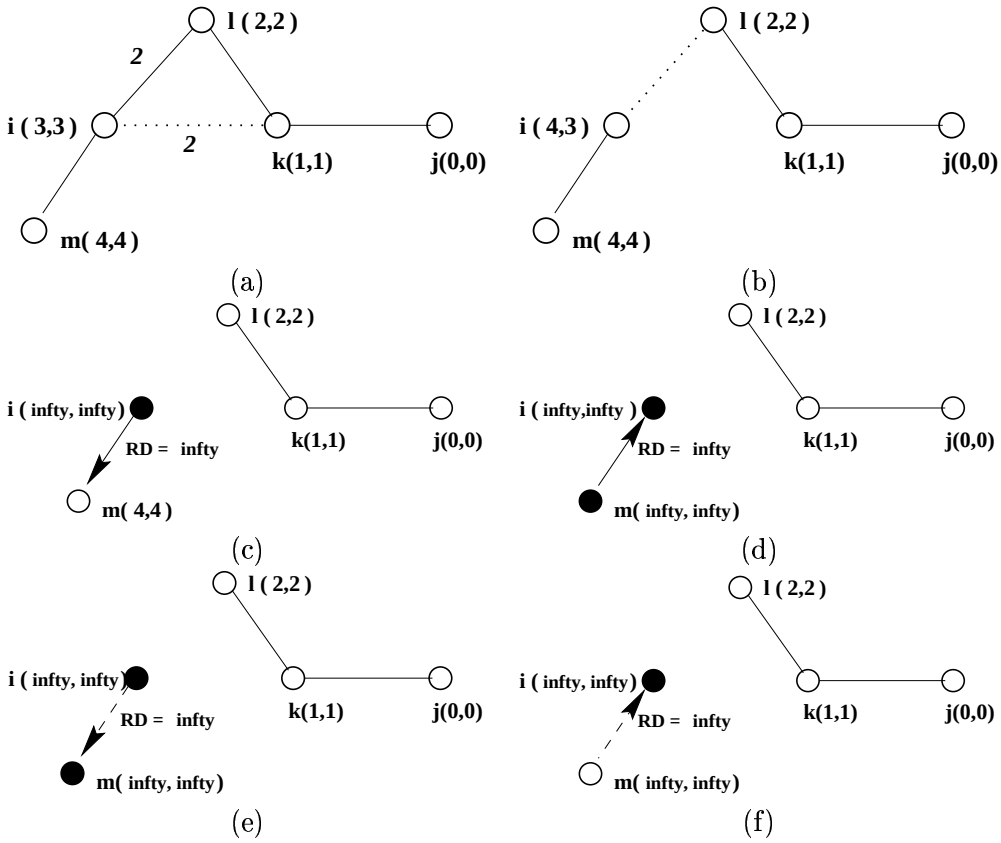


Figure 2.6: Handling partitions due to link failure

router i has no feasible successor. Therefore, it becomes active, sets its distance and feasible distance to infinity and sends a query to m. As shown in Fig. 2.6 (d), when m gets the query, it becomes active because it has no feasible successor. It also sets its distance and feasibility distance to infinity and sends a query to i. Router i sends a reply with infinite distance because it is already active. Since router m has received replies from all its neighbors, it sets its feasible distance to infinity, becomes passive and sends a reply to i. Router i then sets its feasible distance to infinity and becomes passive.

2.7 Deleting Routes

Routes are timestamped when they are entered into the routing table. They are also timestamped whenever data packets for the destination are seen. A timer-driven function compares the timestamp of the route to the current time at the router. If it exceeds the time threshold and if the router is not active for the destination, the route is removed from the routing table.

2.8 Complexity

The performance of ROAM can be measured in terms of the time and communication overhead required to get routing tables to converge and have loop-free paths to the destinations. Actual time is hard to predict because it involves predicting varying inter-router communication time and other delays associated with queuing,

etc. Consequently, we assume that the protocols behave synchronously, which means that all actions are taken by the routers in discrete steps. A router receives its inputs, processes them, makes changes to its routing tables and sends updates all in the same step. The neighboring routers receive the updates in the next step. We start measuring the number of steps and messages after a single topological change. This change could be a link failure, link addition or a link cost change. The neighboring router discovers the change in the first step. During the last step, at least one router receives and processes updates from a neighbor, after which all routing tables are correct and no more updates need to be sent till the next topological change. *Time complexity* measures the number of steps it takes for this process and *communication complexity* measures the number of messages it takes.

In ROAM, a router searches for a destination if the destination is not already in the routing tables. This involves sending a query with an infinite distance for the destination. The query is broadcast to all neighbors. Each neighbor that gets the query checks to see if it has a routing table entry for the destination. If it does not, then the neighbor becomes active and sends a query with infinite distance to all its neighbors which includes the one that sent it the original query. A router that is already active and receives a query does not send any more queries. Thus, one can see that a search query cannot be sent over a link more than twice. Therefore, the communication complexity is $O(|E|)$, where $|E|$ is the number of edges in the network. The time complexity is $O(d)$ where d is the diameter of the network.

After a single link failure or link-cost increase, the time complexity is the

same as the Jaffe-Moss algorithm [26]. In the worst case, all routers upstream of the destination must freeze their routing table entries for the destination. Therefore, the time complexity is $O(x)$, where x is the number of routers affected by the routing table change. The communication complexity is $O(6Dx)$, where D is the maximum degree of the router.

Any router that receives information reporting a distance decrease will always be able to find a feasible successor. Updates are only sent if the distance changes by more than the threshold. Therefore, link additions can at best have no reaction, at worst have a message complexity of $O(2Dx)$ and a time complexity of $O(l)$, where l is the longest path to a destination. To reduce bandwidth used for routing packets, ROAM minimizes the number of update packets sent. In the protocol, update packets have different functionality as compared to query and reply packets. Query packets are sent in two cases, the first being when a source router does not have an entry for a destination router. The second use of query packets is to inform upstream neighbors about the loss of a feasible successor, so that the upstream neighbors can update their distances to the destination and choose successors that offer loop-free paths. Updates are sent to inform neighbors of distance changes in a router when it changes state from active to passive or when it is already passive. We minimize the updates sent by requiring that routers not send updates unless the distance change is greater than a certain pre-specified threshold, i.e, $|D_{old} - D_{new}| > \Delta D$. A simple argument can be used to show that the case where all routers are passive is the worst case for deviation from the optimum path and the amount of deviation is equal to

$\sum_{x=1}^{m} Min(\Delta D, \Delta D_j^{s_x})$, where m is the number of intermediate routers and $\Delta D_j^{s_x}$ is the difference between the best path through a neighbor that is part of the successor set and the best path through a neighbor that is not in the successor set.

2.9 Proof of Correctness

To prove that ROAM is correct, we need to prove that it maintains loop-free paths to all destinations and does not deadlock in any state and converges to the correct distances. Because the successor graphs of different destinations are created and maintained independent of each other, one can prove correctness of the protocol by proving correctness for an arbitrary destination j.

The routers in N, their successors and the links from routers to their successors define a graph that we term $S_j(G)$. For the protocol to be loop-free, this graph has to be a directed acyclic graph at all times. The graph consisting of the routers upstream of router i that become active because of a query sent by router i, is called the *active* acyclic successor graph (ASG) of router i for destination j and is denoted by $S_{ji}(G)$.

Theorem 1 proves that ROAM is loop-free. Lemma 1 proves that no router can remain active after it has received all the replies to its query. Lemma 2 proves that ROAM is loop-free if successors are picked using PFC and AFC. Lemma 4 proves that ROAM is loop-free even in the presence of diffusing computations. Lemma 5 proves that ROAM handles multiple diffusing computations correctly. Theorems 2 and 3 prove that ROAM is live and safe respectively. The assumptions made in the

proofs are the same as the ones delineated in section 2.1.

2.9.1 Loop Freedom

When at time $t = 0$, all the routers are initialized, they have no entries for any other destinations. The graph $S_j(G)$ consists only of all the routers in the graph with no links between them. This graph is trivially loop-free and has correct paths.

Assume that a loop $L_j(t)$ is formed for the first time at t. For a loop to be formed a router i must choose a router upstream from it in $S_j(G)$ as a successor. $L_j(t)$ is formed because a router i changes its successor from b to a due to a change in its distance $D_j^i = D_{jb}^i + l_b^i$ at time t, where b was the successor s_j^i at time t_b and $t_b < t$. Because of the loop $L_j(t)$, $P_{ai}(t) \subset P_{aj}(t)$.

The path $P_{ai}(t)$ consists of the sequence of routers $\{a = s[1, new], s[2, new], ..., s[k, new], ..., i\}$, as shown in Fig. 2.7. The router at the k^{th} hop at time t is $s[k, new]$ and $s[k+1, new]$ is the successor of $s[k, new]$ at time t. The time at which $s[k, new]$ picks $s[k+1, new]$ as its successor is denoted as $t_{s[k+1,new]}$, where $t_{s[k+1,new]} < t$. This is the last time a change was made in the routing table of $s[k, new]$ for destination j. It is seen from the definition that:

$$\begin{aligned} s_j^{s[k,new]}(t_{s[k+1,new]}) &= s_j^{s[k,new]}(t) \\ D_j^{s[k,new]}(t_{s[k+1,new]}) &= D_j^{s[k,new]}(t) \end{aligned}$$

The time at which the last update is sent by $s[k, new]$ to its predecessor $s[k-1, new]$ is denoted by $t_s[k+1, old]$. This is the last update that is sent before

time t. Router $s[k, new]$'s successor at time $t_s[k+1, old]$ is denoted by $s[k+1, old]$ which may or may not be the same as $s[k+1, new]$. The times described above have the following relationship.

$$t_{s[k+1,old]} \leq t_{s[k+1,new]} \leq t$$

Furthermore, it also true that $s_j^{p[i]}(t) = i, s_j^i(t_b) = b$, and $t_b < t$ By definition, $D_j^{*i}(t_i) \leq D_j^i(t_i)$ at any time t_i, and $D_j^{*i}(t_1) \leq D_j^{*i}(t_2)$ if $t_1 < t_2$.

Theorem 1: *ROAM is loop-free at all times.*

Proof: The proof follows from Lemmas 1, 2, 3, 4 and 5. □

Lemma 1: *When a router becomes passive, it must send a reply to its successor if it is not the origin of the diffusing computation.*

Proof: When a router i that is a neighbor of k, receives a query from k and k is not its successor, it immediately sends a reply containing its current distance to the destination. This happens irrespective of router i's being passive or active.

If router i is passive and it receives a query from k and k is its successor, it looks for a feasible successor that satisfies PFC or AFC. If there is a successor that satisfies PFC, then router i sets $o_j^i = 1$ and sends a reply to k. If there is no successor that satisfies PFC , then the router becomes active and tests if any neighbor satisfies AFC. If there is such a neighbor, then router i sets $o_j^i = 1$ and sends a reply to k. If no neighbor satisfies AFC, then router i sets $o_j^i = 3$. Router i forwards the query to all its neighbors, irrespective of a neighbor's satisfying AFC. Therefore, the only

condition where a reply is required is when no neighbor satisfies either AFC or PFC and $o_j^i = 3$.

When router i is active and it receives a query from current successor k, then i has to be the origin of the earlier diffusing computation because k cannot send two consecutive queries without receiving router i's reply. After processing k's query, router i sets $o_j^i = 2$.

When router i becomes passive and $o_j^i = 3$, it sends a reply to its successor and sets $o_j^i = 1$. When router i becomes passive and $o_j^i = 2$, it checks to see if PFC is satisfied. If it is, then the router becomes passive and sends a reply, else it sets $o_j^i = 3$ and we get back to the earlier condition. Therefore, if router i receives a query from a successor, it sends a reply back once it becomes passive. □

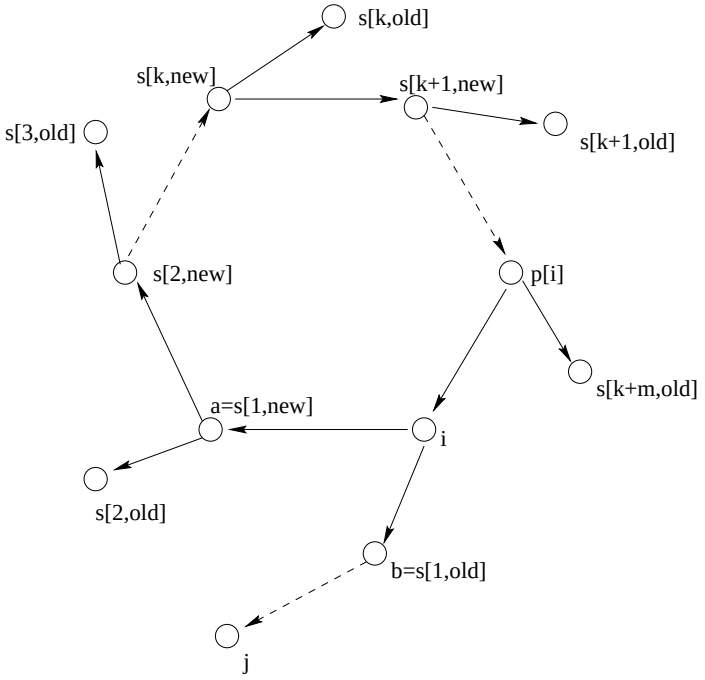


Figure 2.7: Structure of possible loop caused by change of neighbor

Lemma 2: *If routers are always able to pick successors for destination j, using PFC or AFC, then the resulting graph $S_j(G)$ is always loop-free.*

Proof: When at time $t = 0$, all the routers are initialized, they have no entries for any other destinations. The graph $S_j(G)$ consists of all the routers in the graph with no links between them and infinite distances to all destinations. This graph is a disconnected graph made of N components and is trivially loop-free. Assume that a loop $L_j(t)$ is formed for the first time at t. For a loop to be formed a router i must choose a router upstream from it in $S_j(G)$ as a successor. This can only happen if the router that picks up a new successor becomes active sometime before t or at time t, because a router cannot change successors while being passive.

For feasibility conditions to be satisfied, we need to consider two different cases. The first case occurs when the router becomes active because PFC is not satisfied. The router then tests for AFC. If the router does not find any neighbor to satisfy condition $D^i_{jk} < FD^i_j$, then the router keeps its former successor. If the router does find a neighbor that satisfies the AFC, then the router uses that neighbor as the successor while being active. When a router keeps its former successor there is no chance of a loop forming if the original graph was loop-free. Therefore, we only need to investigate the condition where a router is active and finds AFC satisfied. The second case for a change of successor occurs when a router finds a new neighbor that satisfies PFC and makes the neighbor its successor.

If either PFC or AFC have to be satisfied when a router $s[k, new] \in P_{aj}(t)$ makes router $s[k+1, new] \in P_{aj}(t)$ its successor at time $t_{s[k+1,new]}$ it must be true

that

$$D^{s[k,new]}_{js[k+1,new]}(t) = D^{s[k,new]}_{js[k+1,new]}(t_{s[k+1,new]}) < FD^{s[k,new]}_{j}(t_{s[k+1,new]})$$

Since all links costs are positive and either PFC or AFC must be satisfied by every router in $P_{ai}(t)$, we get the following inequalities while traversing it:

$$\begin{aligned}
FD^{i}_{j}(t) = D^{*i}_{j}(t) &> D^{i}_{ja}(t) = D^{a}_{j}(t_{s[2,old]}) \\
D^{a}_{j}(t_{s[2,old]}) \geq D^{*a}_{j}(t_{s[2,old]}) &\geq D^{*a}_{j}(t_{s[2,new]}) \\
&= FD^{a}_{j}(t_{s[2,new]}) > D^{a}_{js[2,new]}(t) \\
&\vdots \\
D^{s[k-1,new]}_{js[k,new]}(t) &= D^{s[k,new]}_{j}(t_{s[k+1,old]}) \\
&\geq D^{*s[k,new]}_{j}(t_{s[k+1,old]}) \\
&\geq D^{*s[k,new]}_{j}(t_{s[k+1,new]}) \\
&= FD^{s[k,new]}(t_{s[k+1,new]}) > D^{s[k,new]}_{js[k+1,new]}(t) \\
&\vdots \\
D^{p[i]}_{ji}(t) &= D^{i}_{j}(t_b) \geq D^{*i}_{j}(t) = FD^{i}_{j}(t).
\end{aligned}$$

The above set of inequalities leads to the erroneous conclusion that $FD^{i}_{j}(t) > FD^{i}_{j}(t)$. Therefore, it follows that no loop can be formed in $S_j(G)$ if the PFC and AFC are used while picking a new successor. □

For the next lemma, we assume that the successor graph $S_j(G)$ was loop-free

before time t and consider the case where a single diffusing computation takes place in graph G.

Lemma 3: *Consider a set of routers that form a path $P_{aj}(t)$. If router $s[k, new]$ is passive at time t, it must be true that*

$$D_{js[k,new]}^{s[k-1,new]}(t) > D_{js[k+1,new]}^{s[k,new]}(t) \tag{2.1}$$

Proof: Consider the passive router $s[k, new] \in P_{aj}(t)$ that joined $P_{aj}(t)$ at time $t_{s[k+1,new]} < t$. This router is the successor of router $s[k-1, new]$, and router $s[k+1, new]$ is in turn its successor at time t.

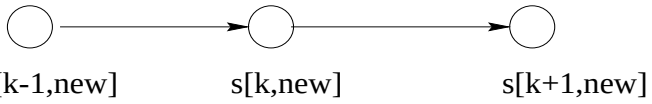


Figure 2.8: Path containing three consecutive routers towards j

Since there was a single diffusing computation, router $s[k, new]$ must have become passive before time t. Router $s[k, new]$ could have either kept its earlier feasible distance $FD_j^{s[k,new]}$ or reset it when it became passive. Consider the case where router $s[k, new]$ did not reset its feasible distance. This case can be further divided into two sub-cases. In the first case router $s[k-1, new]$ does not receive an update containing the distance that $s[k, new]$ has to destination j at time $t_{s[k+1,new]}$. Time $t_{s[k+1],old}$ is the time $s[k-1, new]$ received $s[k, new]$'s last update. This implies that

$$D_{js[k,new]}^{s[k-1,new]}(t) \quad = \quad D_j^{s[k,new]}(t_{s[k+1,old]})$$

Because the feasible distance is the lower bound on the distance and it did not change,

we get

$$D_j^{s[k,new]}(t_{s[k+1,old]}) \quad \geq \quad FD_j^{s[k,new]}(t) > D_{js[k+1,new]}^{s[k,new]}(t)$$

From the above two equations, the lemma is true for sub-case one. Consider sub-case two, where router $s[k-1,new]$ received an update containing the distance that $s[k,new]$ has to destination j at time $t_{s[k+1,new]}$

$$D_{js[k,new]}^{s[k-1,new]}(t) \quad = \quad D_j^{s[k,new]}(t_{s[k+1,new]})$$

Because the feasible distance is the lower bound on the distance and it did not change, we get

$$\begin{aligned} D_j^{s[k,new]}(t_{s[k+1,new]}) &= D_j^{s[k,new]}(t) \\ &\geq FD_j^{s[k,new]}(t) > D_{js[k+1,new]}^{s[k,new]}(t) \end{aligned}$$

Since the lemma is true for this case, it implies that the lemma is true for all cases where the feasible distance is not reset.

Consider the case where the feasible distance $FD_j^{s[k,new]}$ is reset when router $s[k,new]$ joins $P_{aj}(t)$ at time $t_{s[k+1,new]} < t$. Router $s[k,new]$ became active at time $t_k < t$ and became passive at time $t_{s[k+1,new]} < t$. Since there was only a single diffusing computation, there was no distance increase after router $s[k,new]$ became active and therefore $o_j^i = 1$ or 3. At time $t_{s[k+1,new]}$, successor $s[k+1,new]$ offers the shortest path. If AFC is not satisfied, $s[k,new]$ keeps its old successor while active. Therefore,

$$D_j^{s[k,new]}(t) \quad = \quad D_j^{s[k,new]}(t_{s[k+1,new]}) \leq$$

$$D^{s[k,new]}_{js[k+1,old]}(t_{s[k+1,new]}) \quad + \quad l^{s[k,new]}_{js[k+1,old]}(t_{s[k+1,new]}) = D^{s[k,new]}_{j}(t_k)$$

If AFC is satisfied when $s[k, new]$ becomes active, $s[k, new]$ changes its successor to a router we term as $s[k+1, int]$. The time when $s[k+1, int]$ is picked by $s[k, new]$ can be termed as $t_{s[k+1,int]}$. It follows that

$$\begin{aligned} D^{s[k,new]}_{j}(t) &= D^{s[k,new]}_{j}(t_{s[k+1,new]}) \leq \\ D^{s[k,new]}_{js[k+1,int]}(t_{s[k+1,int]}) &+ l^{s[k,new]}_{js[k+1,int]}(t_{s[k+1,int]}) = \\ & D^{s[k,new]}_{j}(t_k) \end{aligned}$$

It follows from the above two equations that when the feasible distance is reset

$$D^{s[k,new]}_{j}(t) \leq D^{s[k,new]}_{j}(t_k) \tag{2.2}$$

Router $s[k, new]$ sends a query to all its neighbors at time t_k and becomes passive only after it receives all replies at $t_{s[k+1,new]}$. Therefore, all neighbors are aware of $s[k, new]$'s distance to the destination at time t_k. However, it may or may not be the case that router $s[k-1, new]$ has processed an update sent by $s[k, new]$ after time $t_{s[k+1,new]}$. If such an update has been processed, then it must be true that

$$\begin{aligned} D^{s[k-1,new]}_{js[k,new]}(t) &= D^{s[k,new]}_{j}(t_{s[k+1,new]}) \\ &= D^{s[k,new]}_{j}(t) > D^{s[k,new]}_{js[k+1,new]}(t) \end{aligned}$$

If such an update has not been processed, then it is true that

$$D^{s[k-1,new]}_{js[k,new]}(t) > D^{s[k,new]}_{j}(t_k)$$

It follows from Eqn. 2.2 that for the case where the update is not processed

$$D^{s[k-1,new]}_{js[k,new]}(t) > D^{s[k,new]}_{js[k+1,new]}(t)$$

Therefore, it is shown that this lemma is true for all cases. □

Lemma 4: *If only a single diffusing computation takes place in G, then $S_j(G)$ is loop-free at every instant.*

Proof: The proof is by contradiction. Assume that $S_j(G)$ is loop-free before time t and has a loop $L_j(t) \in S_j(G)$ for the first time at time t. This loop is caused by some input event that causes router i to change its successor and become the first router that causes a loop. Let router $b = s^i_j$ be the successor of i before time t. For router i to create a loop $L_j(t)$, it must change its successor to $s^i_j = a \neq b$.

There can be two reasons for router i's changing its successor. Either router i finds that router a satisfies AFC when router i becomes active or router i picks router a as its new successor when it becomes passive.

Consider the case when AFC is satisfied and i is active. For router i to become active, it must be true that router b no longer satisfies PFC at time t. This implies that either D^i_{jb} or l^i_b has increased. Consider the case where the change is in D^i_{jb}. Since there is only a single diffusing computation and $S_j(G)$ is loop free until time t, the change that caused the increase of D^i_{jb} cannot cause any router upstream of i in $P_{ap[i]}(t)$ to become active. If the increase was in l^i_b, then i starts the diffusing computation in $P_{ai}(t)$. In either case, at time t, when router i picks router a as its new successor, it is the only router active in $P_{ai}(t)$. Since FD^i_j is not reset when i is

active and AFC is satisfied, it is true that

$$D^i_{ja}(t) < FD^i_j(t) \leq D^i_j(t)$$

Since all routers in $P_{ap[i]}(t)$ are passive, it follows from Lemma 3 that at time t

$$D^i_{ja}(t) > D^a_{js[2,new]}(t) > \ldots > D^{p[i]}_{ji}(t)$$

Time t' was last time router i sent an update to router $p[i]$. Since the feasible distance has not been reset since then, it follows that

$$D^{p[i]}_{ji}(t) = D^i_j(t') \geq FD^i_j(t') = FD^i_j(t)$$

It follows from the above three equations that $D^i_{ja}(t) > D^i_{ja}(t)$, which is a contradiction. Therefore, router i cannot pick a router upstream from it while being active.

Now consider the case where router i is passive when it picks neighbor a as a successor. For router i to have become active at an earlier time $t_i < t$, it has to be true that the successor at that time (b) did not satisfy PFC, i.e $D^i_{jb}(t_i) + l^i_b(t_i) \neq Min\{D^i_j(t_i)\}$ or $D^i_{jb}(t_i) \not< FD^i_j(t_i)$. This implies that either D^i_{jb} or l^i_b had increased from its previous value. Consider the case where the change is in D^i_{jb}. Since there is only a single diffusing computation and $S_j(G)$ is loop free till time t, no router upstream of i can remain active. If the increase was in l^i_b, then i starts the diffusing computation in $P_{ai}(t)$. In either case, at time t, when router i becomes passive, it has to be true that all routers upstream of it in $P_{ai}(t)$ are passive. If FD^i_j was not reset when i became active, it is true that

$$D^i_{ja}(t) < FD^i_j(t) \leq D^i_j(t)$$

Since all routers in $P_{ap[i]}(t)$ are passive, it follows from Lemma 3 that at time t

$$D^i_{ja}(t) > D^a_{js[2,new]}(t) > \ldots > D^{p[i]}_{ji}(t)$$

Time t' was last time router i sent an update to router $p[i]$. If the feasible distance was not reset since then, it follows that

$$D^{p[i]}_{ji}(t) = D^i_j(t') \geq FD^i_j(t') = FD^i_j(t)$$

It follows from the above three equations that $D^i_{ja}(t) > D^i_{ja}(t)$ which is a contradiction.

If FD^i_j was reset when router i became active at time t_i, it is still true that all the routers in $P_{ap[i]}$ are passive at time t. Therefore, it follows from Lemma 3 that

$$D^i_{ja}(t) > D^a_{js[2,new]}(t) > \ldots > D^{p[i]}_{ji}(t)$$

When router i became active at time t_i, it sent out queries to all its neighbors including $p[i]$. Since there are no distance increases after i becomes active, it follows that

$$D^i_{jb}(t) + l^i_b(t) = D^i_{jb}(t_i) + l^i_b(t_i) = D^i_j(t_i) = D^{p[i]}_{ji}(t)$$

It follows from the above two equations that $D^i_{ja}(t) > D^i_{jb}(t) + l^i_b(t)$. However, for router i to pick router a as its successor when i becomes passive, it has to be true that $D^i_{ja}(t) + l^i_a(t) < D^i_{jb}(t) + l^i_b(t)$, which is a contradiction of the earlier statement. Therefore, router i cannot pick a router upstream from it while being part of a diffusing computation and furthermore, in the presence of a single diffusing computation, $S_j(G)$ is loop-free at every instant. □

Lemma 5: *ROAM considers each computation individually and in the proper sequence.*

Proof: Consider the case in which router i is the only router that can start diffusing computations. If router i generates a single diffusing computation, the proof is immediate from Lemma 4. If router i generates multiple diffusing computations, we know that no router in $S_{ji}(G)$ can send a query before it receives all the replies to the query for which it was currently active. Therefore, because all routers in $S_{ji}(G)$ process each input event in FIFO order, and because each router that becomes passive must send an appropriate reply to its successor if it has any (Lemma 1), it follows that all the routers in $S_{ji}(G)$ must process each diffusing computation individually and in the proper sequence.

Consider now the case in which multiple sources of diffusing computations exist in G. Note that once a router sends a query, it must become passive before it can send another query. Hence, a router can be part of only one active ASG started by itself at any one given time. Furthermore, router i can become active only if its feasible successor sends it a query reporting a distance increase. If $k = s_j^i$, then router i is not the origin of the query and it forwards router k's query and becomes part of an ASG. If $k \neq s_j^i$, then router i sends a reply to router k immediately, which means that $S_{ji}(G)$ is not part of the active ASG to which k belongs. Because the active ASGs of G have an empty intersection at any given time, it follows from the previous case that the lemma is true. □.

2.9.2 Liveness and Safety

To prove that ROAM converges to correct routing tables within a finite time, we assume that there is a finite time t after which there are no topology changes. The topology changes of relevance are link cost changes, link failures and link additions. A router failure is modelled as multiple link failures.

The only situation in which a router waits for an external event to complete its computation is when a router is active and expects a reply from a neighbor in order to become passive. The next lemma proves that ROAM is live.

Theorem 2: *ROAM is live.*

Proof: When a router is active and it receives a query, it immediately sends a reply to the query with its current distance to the destination. When a router is passive and it receives a query from a router other than its feasible successor, it immediately sends a reply because the feasibility condition is satisfied. If the router is passive and it receives a query from its feasible successor, it forwards the query to all its neighbors and becomes active. In order to become passive again a router needs to get replies from all its neighbors. Consider a router, say i, that freezes indefinitely because it has not received a reply from one of its upstream neighbors k_1. The router k_1, in turn, is frozen because one of its neighbors k_2 is frozen. One can follow the set of frozen routers till we reach a leaf router. Such a leaf router exists, because the ASG is loop-free at all times and G is finite and every upstream path in the directed acyclic successor graph has to start at leaf routers that have no parents. The leaf router has

to get replies from all its neighbors since it is not the successor of any of them. From Lemma 1 we know that it is impossible for a leaf router to stay active indefinitely. A leaf router will becomes passive and send a reply to its successor. Therefore, no router in the upstream path can remain active indefinitely.

From the above discussion we see that no router waits forever to get a reply from a neighbor and therefore ROAM is live. □

Lemma 6: *The change in the cost or status of a link is reflected in the distance and routing tables of an adjacent router within a finite time.*

Proof: One of our assumptions is that a lower-level protocol gets information about the status of a link within a finite time. This protocol in turn calls a function of the routing protocol that makes a change in the distance table and routing table if necessary. Therefore, this lemma is true. □

Lemma 7: *The number of different values of the shortest distance to each destination in the routing table of each router in G is finite within the time interval* $(0, t)$.

Proof: There can only be a finite set of distinct link costs because there are a finite number of links and a finite number of link-cost changes in the time segment $(0, t)$. Likewise, at time zero, the only destination distance values a router has is the distance to itself. Furthermore, G has a finite number of destinations.

The value of the shortest distance to a given destination stored at any router of G at time t' $(0 \leq t' \leq t)$ can be equal only to the cost of the link to the destination (Lemma 6), or to the sum of a finite distance value stored in the successor router

chosen for the destination plus the cost of the link to that successor router, or to infinity, in which case there is no successor router. Accordingly, there must be a finite number of distinct values that the shortest distance to a destination can take at any given router of G in the time interval $(0, t)$. □

Lemma 8: *Assume that at time t all routers in G are reachable from one another. Then, a finite time after t, no new updates are being transmitted or processed by any router, and the entries corresponding to each destination j in all topology and routing tables are correct.*

Proof: Theorem 2 shows that no router can be active after a finite time $t_f \geq t$, and Lemma 7 shows that the set of values in the routing tables for the distances to destinations is finite within a finite time interval. We define time $t(k)$ to be the time when a passive router whose shortest path to the destination is k hops, has the correct distance to the destination. $D(k)$ is defined as the distance to the destination via the shortest path of k hops. This lemma can be proved using induction on the number of hops k. From Theorem 2 we know that all routers become passive within some time $t' > t$.

Consider the case for $k = 0$. This case is trivially true because, using Theorem 2, we know that a router has to ultimately become passive at some time $t' > t$ and this router always has a correct distance to itself. In the case of $k = 1$, we consider all routers whose shortest path is one hop. These routers have to be the immediate neighbors of the destination. Lemma 6 proves that these routers know the correct link

cost to the destination in a finite time. Therefore, at some time $t_1 > t'$ all $D(1)$'s are correct. From the inductive hypothesis, all $D(k)$'s are correct within a finite time t_k. Consider a router i whose shortest path contains $k+1$ hops. The path of $k+1$ hops can be divided into a path of k hops from the destination to a neighbor m of i and the last link from m to i. We know that m has a shortest path of k hops, or else i would not have a shortest path of $k+1$ hops. From the inductive hypothesis, we know that there is a time $t_k > t$ within which m has its shortest distance. This shortest distance is sent in updates to all neighbors including i. Therefore within a finite time $t_{k+1} > t_k$, router i has the shortest distance to the destination j. Since there is no change in the distance value of i after t_{k+1}, there are no more updates sent after t_{k+1}. Therefore, we see that within a finite time all routers that have a finite number of hops in the shortest path also have the correct shortest distance values. Since all routers are connected and ROAM is loop-free, using Lemma 7 we know the routers have finite and correct routing table values and therefore this lemma is true. □

Lemma 9: *If at time t, a destination j is unreachable from all routers in a component $C \subset N$, then no router in C can terminate with a non-infinite distance to j.*

Proof: Consider a passive router i in component C that has a $D_j^i < \infty$. If i becomes passive with a non-infinite distance to j, it must be true that one of its neighbors became passive with a non-infinite distance to j. We trace the downstream path and since ROAM is loop-free and live and the downstream path cannot end in j, the path must finally end at a router m that has no successor for j but has a finite distance to

j. This is impossible from the specification of the protocol. Therefore, we prove by contradiction that all routers that are partitioned from a destination have an infinite distance for that destination when they terminate. □

Lemma 10: *Assume that at time t at least one router $j \in G$ is inaccessible to a subset of routers in G. Then, a finite time after t, no new update messages with an entry for router j are being transmitted or processed by routers, and the entries corresponding to router j in all topology and routing tables are correct.*

Proof: After time t, G must consist of one or more connected components and a set of zero or more isolated routers. Because an isolated router sets all its routing-table entries to infinity after detecting that it has no neighbors, the proof needs to consider only connected components.

Without loss of generality, consider a connected component C that is disconnected from destination j. From the discussion in Lemma 9, we know that there must exist one or more routers in C that have no successors for j. We define this set of routers as D. Also from the discussion in Lemma 9, we know that from every router in C we can trace a downstream path to one of the routers in D.

Since each router in D detects that it has no successor, it becomes active and sends a query out with distance set to infinity. From Lemma 6 we know that within a finite time after t all routers in D that started a diffusing computation have to become passive. For routers in D to become passive, all routers upstream of them have to become passive. The routers upstream of routers in D includes all routers in C. From

Lemma 9 we know that when routers in C become passive, they set their distances to infinity. Therefore, this lemma is true. □

Theorem 3: *A finite time after time t, no new update messages are being transmitted or processed by routers in G, and all entries in all distance and routing tables are correct.*

Proof: Assume that the transmission of update messages reporting topological changes never ceases or terminates and that there are incorrect values in the routing tables. This implies that there must be at least one row (call it j) of the routing tables for which either an infinite number of update messages are generated or incorrect distance or successor information is obtained. After time t, either all routers are mutually reachable or at least one is inaccessible from a subset of routers in the graph. Therefore, because ROAM treats each destination independently of any other, it follows from Lemmas 6 to 10 there cannot be infinite updates and therefore, this theorem is true. □

2.10 Discussion on uses of ROAM in wired networks

As the use of the Internet increases, we can foresee Internet-supported enterprises in which all business activities are conducted via the Internet. Financial services, securities exchanges and emergency services are examples of applications that will require reliable Internet connectivity. In such situations, it is not unusual for organizations to provide topological redundancy in the form of multiple links with

separate gateway routers to the Internet. One issue with using multiple egress links is that manual configuration of internal routers would be needed to make the default route point to one of the gateway routers. This would require considerable planning and monitoring. Running an on-demand routing protocol in the routers of the network would allow routers to dynamically pick different gateway routers for different destinations in the Internet. This would provide implicit load balancing, because some gateway routers can offer better paths to certain destinations. Also, the routers would transition smoothly to any available gateway router if the currently used link to the Internet failed. The main advantage of on-demand routing over the table-driven routing approach would be that internal routers would have to maintain routes only for the subset of routes they are using. The flood search used by on-demand routing would only be propagated up to the edge of the organization network. This mechanism can be used to maintain routes to both internal destination and external destinations.

2.11 Conclusions

We have presented ROAM, the first on-demand routing algorithm that provides multiple loop-free paths without the need for complete path information, sequence numbers refreshed periodically, or time stamps. We have proved that ROAM is loop-free and converges in a finite time. We also examined the "searching to infinity" problem and eliminated its occurrence in ROAM, such that sources do not send repeated flood searches in the event of destinations being unreachable. We presented the time and communication complexity results for ROAM. ROAM is very applicable

on wired networks, wireless networks with static nodes, and could also be applied to wireless networks with some degree of mobility.

Chapter 3

On demand Routing using Dynamic Source Tracing

In the ROAM protocol, we assumed that reliable updates are available to the routing protocol. Reliability can be added at the link layer. However, most existing wireless stack implementations do not have reliable link layers. On-demand routing protocols adapted for unreliability use source routed data packets or time stamps to validate updates. Protocols using timestamps are susceptible to inefficient behavior in the presence of node failure (timestamp getting reset to an arbitrary value). Source routing is not an efficient option as the size of the network increases. Eventually, the size of the headers will be more than the data, especially in the proposed IPv6 where an address has 128 bits.

In this chapter, we introduce and analyze the DST (dynamic source tree) protocol [43, 44], which constitutes a new approach for on-demand distance vector

routing for ad-hoc networks with unreliable delivery. DST uses a source-tracing algorithm similar to the one advocated in prior table-driven routing protocols in which routers maintain routing information for all network destinations [36]. To reduce the number of loops, the source-tracing algorithm allows for complete paths to be checked for loops before being added to the routing table. DST uses information about the length and second-to-last hop (*predecessor*) of the shortest path to all known destinations thus eliminating the counting to infinity problem of DBF.

A node running DST maintains shortest paths to all required destinations in its routing tables. A node also maintains the routing tables of all its known neighbors. A node uses the routing tables of known neighbors along with the link costs to known neighbors to generate its own routing table. A routing message broadcast by a node contains a vector of entries where each entry corresponds to a route in the routing table; each entry contains a destination identifier j, the distance to the destination D_j^i and the second-to-last hop to that destination p_j^i. The routes are acquired by flooding queries through the network and replies are sent back by routers that know of the destination.

In a MANET network, a node principally consists of a router, which may be physically attached to multiple IP hosts (or IP-addressable devices). Instead of having interface identifiers, a router has a single node identifier, which helps the routing and other application protocols identify it. In a wireless network, a node has radio connectivity with multiple nodes using a single physical radio link. Accordingly, we map a physical broadcast link connecting a node and its multiple neighbors into point-

to-point links between the node and its neighbors. Each link has a positive cost associated with it. If a link fails, its cost is set to infinity. A node failure is modelled as all links incident on the node getting set to infinity.

For the purpose of routing table updating, a node A considers another node B as its neighbor if A receives an update with distance zero to B. Node B is no longer node A's neighbor when node A cannot send data packets to it.

DST is designed to run on top of any wireless medium-access protocol. Routing messages are broadcast unreliably and the protocol assumes that routing packets may be lost due to changes in link connectivity, fading or jamming. However, it is assumed that a lower-level protocol can inform the routing protocol when data packets cannot be unicast to the next hop. Therefore, DST needs no link-layer protocol for monitoring link connectivity with neighbors or transmitting reliable updates. If such a layer is provided, then DST can be made more proactive by identifying lost neighbors before data for them arrives. Having node connectivity information will result in faster convergence time and will decrease loss of data packets.

3.1 Routing Information maintained in DST

A router in DST maintains a *routing table*, a *distance table*, a *data buffer* and a *query table*.

The set of known destinations is denoted by N and N_i denotes the list of known neighbors.

The routing table at router i contains entries for all known destinations. Each

entry consists of the destination identifier j, the successor to that destination s_j^i, the second-to-last-hop to the destination p_j^i, the distance to the destination D_j^i and a route tag tag_j^i. When the element tag_j^i is set to *correct*, it implies a loop-free finite value route. When it is set to *null*, it implies that the route still has to be checked and when it is set to *error*, an infinite metric route, or a route with a potential loop, is implied.

The distance table at router i is a matrix containing, for each $k \in N_i$ and each destination j, the distance value of the route from i to j through k, D_{jk}^i and the second-to-last hop p_{jk}^i on that route. D_{jk}^i is always set equal to $RD_j^k + l_k^i$, where RD_j^k is the distance reported by k to j in the last routing message and l_k^i is the link cost of link (i, k). The link cost may be set to one to support minimum-hop routing, or it may be set to some other link parameter like latency or bandwidth.

The data buffer is a queue that holds all the data packets waiting for routes to destinations. There are various approaches for buffer management. However, we chose to use the scheme used by most existing on-demand routing protocols. The buffer has a limited size and if it fills up, the packet at the head of the buffer is dropped to make room for the incoming data packet. Each data packet also has a time value, which is set to the time when the packet is put into the buffer. A packet that has been in the buffer for more than *data_pkt_timeout* seconds is dropped. The data buffer is checked periodically for any packets that may be sent or dropped.

The query table is used to prevent queries from being forwarded indefinitely. We use a scheme similar to the one used in DSR, which allows for two kinds of queries: (a) queries with a zero hop count, which are propagated to neighbors

only; and (b) queries with maximum hop count, which are forwarded to a maximum distance of MAX_HOPS hops from the sender. For each destination j, the query table contains the last time a maximum hop query was sent qs_j^i, the last time a zero hop query was sent zqs_j^i, the hop count of the last query sent hqs_j^i, the last time a query was received qr_j^i. At the source of the flood search, two maximum hop count queries are always separated by *query_send_timeout* seconds. A query is forwarded by a receiver only if the difference between the time it is received and qr_j^i is greater than *query_receive_timeout*, where *query_receive_timeout* is slightly less than *query_send_timeout*. The reasoning for this can be explained using Fig. 3.1. In the figure, times $t1$ and $t3$ correspond to times when the querying is started at the source and $t3 - t1 \geq$ *query_send_timeout*. Since it is possible for the queries to travel different paths, we can have a condition where the first flood took a longer time to reach the forwarding node than the second flood, i.e., $(t2 - t1) \geq (t4 - t3)$. If *query_receive_timeout* were equal to *query_send_timeout*, the second flood will not be propagated. However, we require the *query_receive_timeout* to be large enough to prevent propagation of queries from the same flood search. This is the first protocol to use only local clocks to separate flood searches and this is an important advantage over using sequence numbers, because this makes the protocol impervious to node failures.

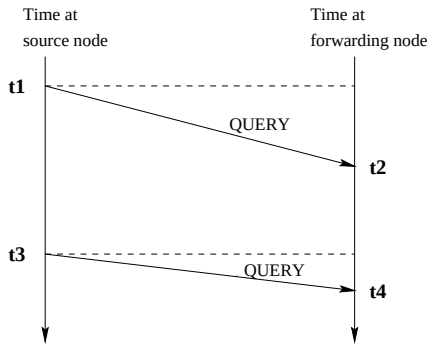


Figure 3.1: Querying timeline at source and forwarding nodes

3.2 Routing Information exchanged in DST

There are two types of control packets in DST - *queries* and *updates*. All control packets headers have the source of the packet ($pkt.src$), the destination of the packet ($pkt.dst$), the number of hops ($pkt.hops$) and an identifier $pkt.type$ that can be set to $QUERY$ or $UPDATE$. Each packet has a list of routing entries, where each entry specifies a destination j, a distance to the destination RD_j^i and a predecessor to the destination rp_j^i.

If the MAC layer allowed for transmission of reliable updates with no retransmission overhead (which is the case of wireless MAC protocols presented in [48, 53]), incremental routing updates would suffice to update routing tables among neighbors. In this study, however, we assume a MAC protocol based on collision avoidance. In order to avoid collisions of data packets with other packets in the presence of hidden terminals, such protocols require nodes to defer for fixed periods of time after detecting carrier [18, 23]. Accordingly, sending larger control packets does not decrease throughput at the MAC layer, because the overhead ($RTS - CTS$ exchange) for the

MAC protocol to acquire the channel does not depend on packet size. Therefore, in the rest of this paper, we assume that routers transmit their entire routing tables when they send control messages. Control packet size may affect the delay experienced by packets in the MAC layer. However, as our simulations show, this does not affect data packet delays because the number of control packets we generate is substantially low.

Data Packets in DST only need to have the source and destination in the header.

3.3 Creating Routes

When a network is brought up, each node (i) adds a route to itself into its routing table with a distance metric (D_i^i) of zero, the successor equal to itself (i) and the tag (tag_i^i) set to correct. To differentiate a route to itself from all other routes, a node sets the local host address (127.0.0.1) as the predecessor to itself.

When a data packet is sent by an upper layer to the forwarding layer, the forwarding layer checks to see if it has a correct path to the destination. If it does not, then the packet is queued in the buffer and the router starts a route discovery by broadcasting queries. Route discovery cycles are separated by *query_receive_timeout* seconds. One zero hop query and one maximum hop query are sent in every cycle. A zero hop query allows the sender to query neighboring routing tables with one broadcast. If the zero hop query times out ((present time - zqs_j^i) > *zero_qry_send_timeout*), then an unlimited hop query (with *pkt.hops* set to MAX_HOPS) is sent out. Consider the six-node network in Fig. 3.2.a where all link costs are of unit value and where

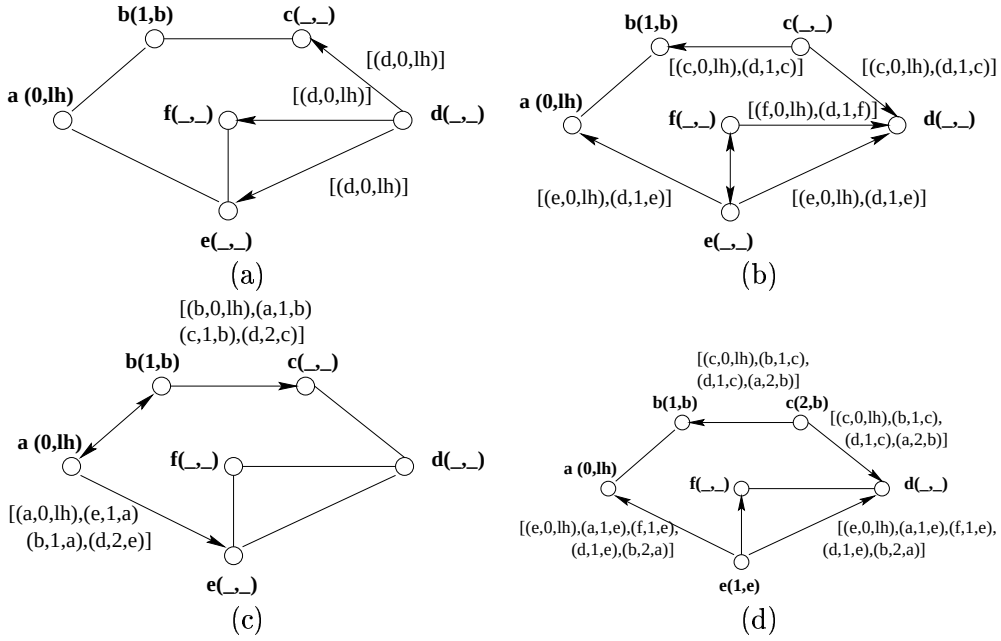


Figure 3.2: Example of the Query-Reply process in DST. Node d is searching for destination a. The parenthesis contains the distance and predecessor values for a.

node d broadcasts a query for destination a , with the $pkt.src$ set to d, $pkt.dst$ set to a, and $pkt.hops$ set to MAX_HOPS. The parenthesis next to each node in the example depicts the routing table entry (distance, predecessor) for destination a. The symbol lh stands for local host address (127.0.01). The query packet contains a list of all the routing table entries of the sender d. The entries are shown within the square brackets, each entry in the (destination, distance, predecessor) form. The entries are in a increasing-distance order, such that a node i receiving a query from an unknown neighbor k, adds the neighbor k to its distance tables on reading the first entry in the query and proceeds to consider all other entries as the distances reported by k.

Consider node e, where the query from d is received. To process the query, each entry (j, RD_j^d, rp_j^d) is read (procedure *Query* in Fig. 3.3). If the entry is

for an unknown destination, then the destination is initialized ($D_j^i \rightarrow \infty$, $p_j^i, s_j^i \rightarrow NULL\_ADDR$; $D_{jk}^i \rightarrow \infty$, $p_{jk}^i \rightarrow NULL\_ADDR \; \forall k \in N_i$. Then, the distance table entry for neighbor d is updated in the procedure *DT_Update* (Fig. 3.3). Since the distance RD_d^d is equal to zero, d is marked as a neighbor. The procedure *DT_Update* (Fig. 3.3) also updates the value for j reported by other neighbors whose path contains d. This step helps prevent permanent loops by preemptively removing stale information.

Finally, procedure *RT_Update*(Fig. 3.4) is called to update routing table entries; this procedure iterates through each known destination and picks the neighbor k as a successor to destination j if both the following conditions are true

1. k offers the shortest distance to all nodes in the path from j to i.

2. the path from j to k does not contain i and does not contain any repeated nodes.

If either of the two conditions are not satisfied, then tag_j^i is set to *error*. Else, it is set to *correct* and neighbor k is designated the successor and the distance value to j is set to D_{jk}^i and the predecessor is set to p_{jk}^i.

After processing all entries and updating the routing table, node e checks to see if it has a route to a. Since there is no route, a query packet is broadcast with the same header fields as the processed query, besides *pkt.hops* which is decremented by one if all of the following conditions are met

1. Node does not have a route to *pkt.dst*

2. *pkt.hops* is greater than one

3. The time elapsed since the last query was received is greater than *query_receive_timeout*

The routing entries added to the forwarded query reflect the routing table entries of current node e. The packet is then broadcast to the limited broadcast address. In Fig. 3.2.b, nodes e, f and c broadcast queries.

In Fig. 3.2.c, we see that nodes e, f, and a do not send any more queries, because the time elapsed since the last query sent is shorter than *query_receive_timeout*. On the other hand, at nodes a and b, a finite and valid route to a is found and a reply update is sent. A reply update sent by a node i has a different structure than a regular update, which has $pkt.dst$ set to the limited broadcast address and $pkt.src$ set to i. The reply update sent by b has field $pkt.dst$ set to the $pkt.src = d$ of the query and the field $pkt.src$ set to the $pkt.dst = a$ of the query. All updates are broadcast to the limited broadcast address.

When node i receives an update, it checks the value of $pkt.dst$. If the value is other than the limited broadcast address, then the update being sent is a reply update, else it is a regular update. As shown in procedure *Update* of Fig. 3.3, the entries are processed in a manner similar to the entries of the query. A regular update is broadcast in response to a regular update, with $pkt.dst$ set to the limited broadcast address and $pkt.src$ set to i if any of the two following conditions is satisfied

1. Distance to a known destination increases.

2. Next hop to a known destination changes.

3. A node loses the last finite route to a destination.

The reply update has different rules for propagation. In Fig 3.2.d , a reply update is rebroadcast by e with the original $pkt.dst$ and $pkt.src$, because the following conditions are met

1. A finite path to $pkt.dst = d$ exists.

2. Distance to $pkt.src = a$ changes from infinite to finite after processing the reply update.

3. The reply update is received from a neighbor whose distance to $pkt.dst = d$ is greater than e's.

Nodes a and b do not rebroadcast reply updates, because the second condition is not satisfied. Node d obtains a reply update from node e and sets its successor to node e after processing the entries in the query. Node d does not send any more reply updates. However, a regular update is sent if any of the conditions for regular updates are satisfied.

Using the above procedure, DST allows a source to obtain multiple paths to a required destination. By forwarding a reply update only when the route to the required destination changes from infinite to finite, the number of updates is reduced at the expense of non-optimal routes. The same reasoning motivates not sending regular updates when a new destination is found or when a distance to a destination reduces. However, distance increases and next-hop changes prompt updates because a loop can occur *only* when a node picks as successor a neighbor that has a distance greater than itself.

```
Procedure Add_Nbr(k)
called when node i learns of new neighbor k
begin
   N_i ← N_i ∪ k
   for all (j ∈ N)
      D^i_jk ← ∞
      p^i_jk ← NULL_ADDR
   end for all
end

Procedure Rmv_Nbr(k)
called when node i learns of loss of neighbor k
begin
   N_i ← N_i − k
   for all (j ∈ N)
      tag^i_j ← null
   send ← FALSE
   RT_Update(send)
   if (send = TRUE)
      send update with source(i) and destination(BDCAST_ADDR)
end

Procedure DT_Update(k,j,RD^i_j,rp^i_j)
updating distance table entry
begin
   if (RD^i_j < ∞)
      D^i_jk ← RD^i_j + 1
   else D^i_jk ← ∞
   p^i_jk ← rp^i_j
   for all (b ∈ N_i)
      if k is in path from i to j via b
         D^i_jb ← D^i_kb + RD^i_j
   end for all
end

Procedure Query(pkt,nbr)
called for processing query
begin
   for each entry (j, RD^i_j, rp^i_j) in pkt
      if (j ∉ N)
         if (RD^i_j = ∞)
            continue
         else
            initialize j
            if (RD^i_j = 0)
               Add_Nbr(j)
         end else
      end if
      else
         if (RD^i_j = 0 and j ∉ N_i)
            Add_Nbr(j)
      end else
      DT_Update(pkt.src, j, RD^i_j, rp^i_j)
   end for each
   send ← FALSE
   RT_Update(send)
   if (tag^i_pkt.dst = correct)
      send update with source(pkt.dst) and destination(pkt.src)
   else
      if ( present time - qr^i_pkt.dst > query_receive_timeout)
         if (pkt.hops > 1)
            send query with destination(pkt.dst), hops (pkt.hops − 1))
            and source(pkt.src)
         if (pkt.hops ≥ 1)
            qr^i_pkt.dst ← present time
      end if
   end else
end

Procedure Recv_Ctl_Packet(pkt, nbr)
when node i receives a control packet from nbr
begin
   if (pkt.type = QRY)
      Query(pkt,nbr)
   else
      if (pkt.dst = BDCAST_ADDR)
         Update (pkt,nbr)
      else
         if(pkt.dst ∈ N and tag^i_pkt.dst = correct)
            Update (pkt,nbr)
   end else
end

Procedure Update(pkt,nbr)
called for processing update
begin
   newpath ← FALSE
   if (pkt.dst ≠ BDCAST_ADDR)
      if (pkt.src ∉ N or tag^i_pkt.src ≠ correct)
         newpath ← TRUE
   for each entry (j, RD^i_j, rp^i_j) in pkt
      if (j ∉ N)
         if (RD^i_j = ∞)
            continue
         else
            initialize j
            if (RD^i_j = 0)
               Add_Nbr(j)
         end else
      end if
      else
         if (RD^i_j = 0 and j ∉ N_i)
            Add_Nbr(j)
      end else
      DT_Update(pkt.src, j, RD^i_j, rp^i_j)
   end for each
   send ← FALSE
   RT_Update(send)
   if (pkt.dst = BDCAST_ADDR)
      if (send = TRUE) then send update source(i) and
         and destination(BDCAST_ADDR)
   else
      if (pkt.dst = i)
         if (send = TRUE) then send update source(i) and
            and destination(BDCAST_ADDR)
      else
         if (newpath = TRUE and (pkt.src ∉ N or tag^i_pkt.src ≠ correct))
            newpath ← FALSE
         if (tag^i_pkt.dst = correct and newpath = TRUE
            and pkt.src is not in the path to pkt.dst)
            send update with source(pkt.src) and destination(pkt.dst)
         else
            if (send) then send update source(i) and
               and destination(BDCAST_ADDR)
      end else
   end else
end
```

Figure 3.3: Specification of selected procedures in DST

```
Procedure RT_Update(send)
updating routing table entries
begin
   for all (j ∈ N)
      if (j = i)
         continue
      DTMin ← Min{D^i_jb ∀b ∈ N_i}
      if (D^i_{j s^i_j} = DTMin) then ns ← s^i_j
      else ns ← b|{b ∈ N_i and D^i_jb = DTMin}
      x ← j
      loop ← FALSE
      for (m = 0; m < |N|; m++)
         visited[m] ← NULL_ADDR
      num_visited ← 0
      while ((D^i_{x ns} = Min{D^i_xb ∀b ∈ N_i})
         and D^i_{x ns} < ∞ and tag^i_x ← null and loop = FALSE)
         m ← 0
         while (m < num_visited)
            if (visited[m] = x or x = i)
               loop ← TRUE
         end while
         x ← p^i_{x ns}
      end while
      if (loop = FALSE and (p^i_{x ns} = IP_LOCALHOST or tag^i_x = correct))
         tag^i_j ← correct
      else
         tag^i_j ← error
      if (tag^i_j = correct)
         if (D^i_j < DTMin) then send ← TRUE
         D^i_j ← DTMin
         s^i_j ← ns
         if (D^i_j = 1) then p^i_j ← i
         else p^i_j ← p^i_{j ns}
      end if
      else
         if (D^i_j ≠ ∞) then send ← TRUE
         p^i_j ← NULL_ADDR
         s^i_j ← NULL_ADDR
         D^i_j ← ∞
      end else
   end for all
end
```

Figure 3.4: Specification of selected procedures in DST (contd.)

3.4 Maintaining Routes

DST does not poll neighbors constantly to figure out link connectivity changes, which avoids control overhead due to periodic update messages, but may result in sub-optimal routes and longer convergence time. A link to a neighbor is discovered only when a update or a query is received from that neighbor. On finding a new neighbor k the neighbor is added to the distance table. (procedure *Add_Nbr* (Fig. 3.3)) An infinite distance to all destinations through k is assumed, with the exception of node k itself and any destinations reported in the received routing message.

A failure of a link is detected when a lower level protocol sends an indication that a data packet can no longer be sent to a neighbor. The procedure *Rmv_Nbr* (Fig. 3.3) is called to remove the neighbor from the distance tables. Then, the procedure *RT_Update* is called to recompute distances to all destinations. Consider the six-node network in Fig. 3.5.a which is the same as that in Fig. 3.2 after the route discovery cycle started by node d for node a is done.

Consider Fig. 3.5.b, in which the link between a and e fails. Node e does not pick any of its neighbors f and d as successors because tracing the path in *RT_Update* allows node e to conclude that it lies in the paths of both f and d towards a. Thus, counting to infinity is avoided by the source tracing algorithm. Since there has been a distance increase, node e broadcasts an update. In Fig. 3.5.c, node d picks node c as its successor and changes its distance to 3 and predecessor to b. Node d sends out a regular update because it increased its distance. Node f also sends an update, which we do not show for the sake of brevity.

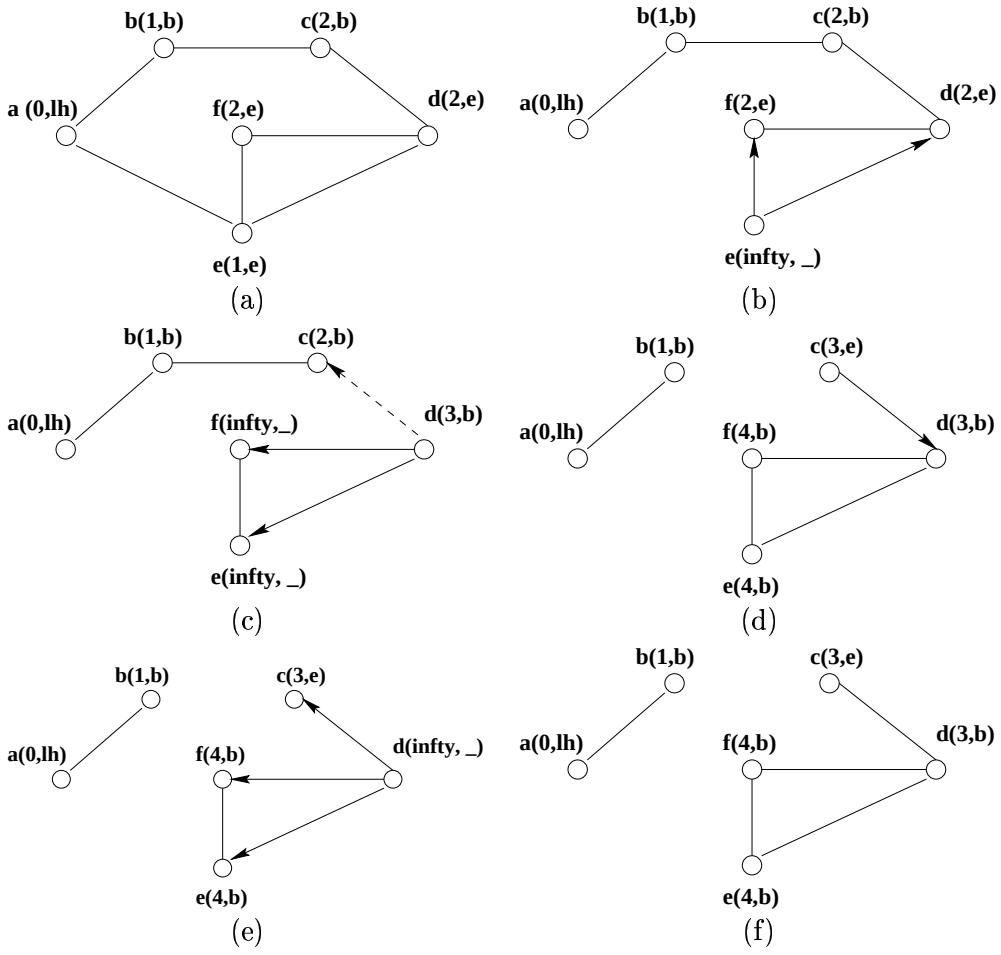


Figure 3.5: Maintaining routes in DST. The parenthesis contain the distance and predecessor values for destination a

Let us assume that, due to some outside interference or fading, node c does not receive node d's update. Meanwhile, in Fig. 3.5.d, the link between c and b fails. Because node c's distance tables reflect a path through node d with predecessor e, node c increases its distance to 3 and changes its predecessor to e. Node c then sends an update. We consider two different sets of events that could happen. In Fig. 3.5.e, the update from b reaches d and d changes its distance to infinity and send out updates which cause e, f and c to reset their distance to a to infinity. In Fig. 3.5.f, we consider the case where the update from node c to node d is lost. This results in node d picking c as successor and node c picking d as its successor, which implies a 2-hop loop in the tables of c and d. To prevent such situations, we provide two conditions that prevent data packets from looping. A data packet is dropped and a regular update is sent if

A. The data packet is sent by a neighbor that is in the path from the present node to the destination of the data packet.

B. The path implied by the neighbor's distance table entry is different from the path implied in the routing table.

If node c receives a data packet from node d for destination j, it drops the data packet, because node d is in it's path to j and sends a regular update. Node d eventually receives and update from c and resets its tables. Thus, DST avoids long-term loops.

3.5 Removing Unused Routes

Each routing table entry has a timestamp associated with it. When a data packet arrives at node i, all the timestamps of the routes corresponding to the path

from i to $pkt.dst$ are updated to the present time.

Periodically, the routing table is checked for routes that have not been used in a sufficiently long time. If that is the case, then the destination is removed from the routing and the distance tables.

3.6 Packet Forwarding

The data packet header contains only the source and the destination of the data packet. When a data packet originated at a node arrives at its forwarding layer, the packet is buffered if there is no finite route to the destination. The node then starts the route discovery process. If a finite and correct route is found, then the packet is forwarded to the successor as specified by the routing table.

If a data packet is not originated at a node, then the data packet is only buffered if there is no entry in the routing table for $pkt.dst$. In this case, route discovery is started by the intermediate node. If there is a correct and finite route then the data packet is first checked for conditions A and B described in Section 3.4. If the two conditions are satisfied, the data packet is forwarded to the successor $s^i_{pkt.dst}$. If there is route with infinite distance, then the packet is dropped and a regular update is broadcast to all neighbors. Eventually, the source of the data will learn of the loss of routes and it will restart the route discovery process.

3.7 Proof of correctness of the DST algorithm

In this section, we show that the basic routing algorithm used in DST is correct for the routes along which data packets flow. We call such a route as an *in-*

use route. All the links in such a route are termed as *in-use links*. The following assumptions are made about the behavior of the routers and the network.

- The set of destinations for which the router has packets are static and exist in the network.
- The link layer can inform the routing protocol within a finite time after the link fails.
- Control packets are exchanged reliably.

We use the following notation to aid the proof

l_j^i: Link cost for link (i, j).

D_j^v: Distance in the routing table entry for destination j at router v.

D_{jk}^v: Distance in the distance table entry for neighbor k for destination j. This can also be defined as the distance from router v to destination j through neighbor k.

s_j^v: Successor (next-hop) in the routing table for destination j which is picked according to the path selection rules of DST.

p_j^v: Predecessor (second to last hop) in the routing table entry for destination j at router v.

p_{jk}^v: Predecessor in the distance table entry for neighbor k for destination j.

Definition 1

The link cost l_j^i for link (i, j) can be extracted from the distance table at router v if there is a column in the distance table for a neighbor k such that $l_j^i = D_{jk}^v - D_{ik}^v$ and

$p^v_{jk} = i$. Similarly, the link weight can be extracted from the routing table as $D^v_j - D^v_i$, where $p^v_j = i$.

Lemma 1

If a routing table is generated based on the rules of DST, any link cost that can be extracted from this routing table can be extracted from a column of the distance table.

Proof

Let N_v denote the neighbors of router v. Let l^i_j be a link cost extracted from the routing table of node v. One of the required conditions for updating distance D^v_j, successor $s^v_j = k$ and predecessor p^v_j is that the distance table entry for k indicates the shortest distance to all nodes in the implied path to j. Therefore, we know that if $s^v_j = k$, then $D^v_j = D^v_{jk}$ and $D^v_i = D^v_{ik}$, which implies that the link cost can be extracted from the distance table. Therefore, Lemma 1 is true. □

Lemma 2

The link cost change of a in-use link will be reflected within a finite time in the routing and distance tables of a neighboring router.

Proof

An in-use link can either go down or suffer a link cost change. If an in-use link goes down, then the MAC protocol will inform the routing protocol within a finite time because the data packets will not get through on that link. Similarly, if the MAC protocol is monitoring the cost of the link, then the MAC protocol will inform the

routing layer of the link cost change. In response to a loss of neighbor the routing protocol will delete the column entry corresponding to the neighbor from the distance table. In response to a link cost change, the link cost is marked in the distance table. In both cases, the change in link cost triggers a reevaluation of the routing table using the procedure *RT_Update*. Therefore, this lemma is true. □

Theorem 1

If the distance entries in the distance table and routing table are finite, then a path can be extracted from the distance and routing table entries and this extracted path is loop-free.

Proof

Fig. 3.6 depicts a permanent loop caused when node a shifts its successor from node g

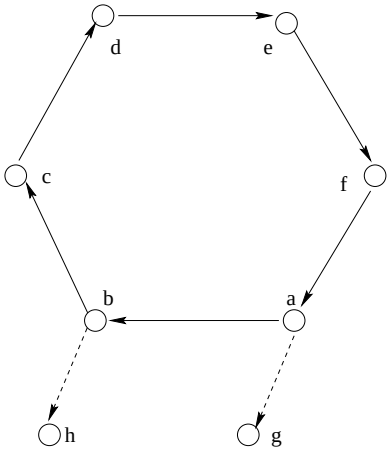


Figure 3.6: Figure depicting a permanent loop

to node b due to a distance change at node a at time T. The head of the arrow in the path is the successor of the tail of the arrow. Without loss of generalization we assume

that the route is for a certain destination j. Assume, at some time before T, node b had switched successors from h to c. There are two cases that we need to consider. In the first case, node b had suffered a distance increase and therefore informed its neighbors, including a of the new path it was using. Since there was no loop before time T, we can make the assumption that $b, c, d, e, f, a, g...j$ was a simple path. In the second case, node b picked node c because link (b, c) decreased in cost, thus making distance through node c shorter than distance through h. Since node b changed a successor, node b would have to inform node a of its new simple path $b, c, d, e, f, a, g...j$. At time T when node a needs to look for a new path, it steps through the path implied if it is used b. This path would be $a, b, c, d, e, f, a, g...j$, which would trigger the condition that node should not accept a path that includes itself as an intermediate node, thus preventing a loop. Therefore, theorem 1 is true. □

Theorem 2

Following a link cost change, there can only be a finite number of updates generated for that change.

Proof

Suppose the link cost change happens at time T. If the link cost decreases, then no update is sent because there is no distance increase for any destination. Therefore, for this case the theorem is true.

Next we consider the case where the link cost increases or the link goes down. If the link is an in-use link, using lemma 2, we know that the change is recorded in

a node adjacent to the link. If the link is not an in-use link, no update is triggered. Since the link is part of the a in-use route, then we know that the change of this link cost creates a distance increase for a destination for which there is data flowing. This causes the node adjacent to the link to send out a single update. A node that receives the update will send out further updates only if the update arrives from the successor to a destination, else the node just processes the update and changes its distance table and sends no more updates. If the update arrives from a successor, then a further update is sent under the following conditions

1. The update increases the distance to a destination.

2. The update causes a change of the successor.

Without loss of generalization we consider updates for a single destination. In Theorem 1, we proved that the finite paths extracted from the routing table entries are loop-free. Let us consider such a path to destination j at node v. Consider the case where node v sends an infinite number of updates. This is only possible because its present successor k in the path is sending it an infinite number of distance increase updates or updates that cause change of successor. Node k can send infinite updates only if its successor in turn is sending infinite updates. If we step through the loop free path to j, we finally end up at node j, which cannot be sending infinite updates for itself. Therefore, by contradiction, this theorem, is true. □

Theorem 3

Within a finite time after the failure of a link (i,k) in the network at time T, all routers using link (i,k) will stop assuming link (i,k) exists.

Proof

If node i is not using link (i, k) to reach any destination j at time T, i.e., (i, k) is not an in-use link, then none of the nodes using i as a forwarding node will be using link (i, k). Therefore, no updates needs to be sent or processed after the event. Suppose node i is using link (i, k) and it fails. This will cause a distance increase at node i because link (i, k) was part of the shortest path of node i to destination j. Hence, node i will send out an update with the new path not containing link (i, k). This update will cause further updates until it reaches a node where both of the cases are true:

- The link (i, k) is not part of the path implied in the routing table.
- The update from the neighbor neither causes a distance increase nor causes the node to switch a successor.

Because the link layer can detect a link failure within a finite time and a node creates and processes an update within a finite time and all paths containing (i, k) are finite loop-free paths, all nodes will stop assuming link (i, k) exists within a finite time. □

Lemma 3

If a node does not have a path to destination j and has data packets for destination j, it obtains a path to j within a finite time after sending a query.

Proof

A router v initiates a route discovery process by sending a zero hop query. If none of the neighbors has any path, a maximum hop query is sent which can traverse the diameter of the network. When a router receives a query for node j, it creates a loop-free path to the source v of the query and reports this path in all future control packets including the query it propagates. Hence, every node that forwards the query knows what path to use to send the reply update to the source v. Because the network is a finite size and is connected, at least one node (i.e., the node j itself) should be able to send a reply. Since all the intermediate nodes have a path to the source, the reply propagates back towards v and all intermediate nodes including v create paths to the destination j. Therefore, a path can be found within finite time. □

Theorem 4

If a path to a node breaks due to link failure and there exits an alternate path, DST finds it within a finite time.

Proof

From Theorem 3 we have that all nodes stop whose route to a destination includes a failed link will stop using that route within a finite time. These nodes will either have an alternate path through some other neighbor or will lose all paths to the destination. If there is an alternate loop-free path, the nodes will choose it. If not then in response to data for the destination, these nodes will start a querying cycle and we know from Lemma 3 that these nodes will get a path within a finite time, if the node still exists. □

Theorem 5

If a node that is a destination for a data flow becomes disconnected, every node that has a path to it will have no path to it within a finite time.

Proof

Every node failure can be considered to be equivalent to multiple link failures. Therefore, using Theorem 3, we can say that after a node failure every node wishing to reach the failed node will have no path. □

3.8 Performance Evaluation in CPT

We ran simulations for three different experimental scenarios to compare DST's average performance against the performance of DSR. These simulations allowed us to independently change input parameters and check the protocol's sensitivity to these parameters. Both the protocols are implemented in *CPT*, which is a C++ based toolkit that provides a wireless protocol stack and extensive features for accurately simulating the physical aspects of a wireless multi-hop network. The protocol stack in the simulator can be transferred with a minimal amount of changes to a real embedded wireless router. The stack uses IP as the network protocol. The routing protocols directly use UDP to transfer packets. The link layer implements the IEEE 802.11 standard [3] and the physical layer is based on a direct sequence spread spectrum radio with a link bandwidth of 1 Mbit/sec.

[htbp]

Table 3.1: Constants used in DSR simulation

Time between ROUTE REQUESTS (exponentially backed off)	500(msecs)
Size of source route header carrying carrying n addresses	$4n$+4(bytes)
Timeout for Ring 0 search	30(msecs)
Time to hold packets awaiting routes	30 secs
Max number of pending packets	50

[htbp]

Table 3.2: Constants used in DST simulation

Query send timeout	5(secs)
Zero query send timeout	30(msecs)
Data packet timeout	30 secs
Max number of pending packets	50
Query receive timeout	4.5 (secs)
MAX_HOPS	17

To run DSR in CPT, we ported the DSR code available in the *ns*2 [17] wireless release. There are two differences in our DSR implementation as compared to the implementation used in [7]. Firstly, we do not use the *promiscuous* listening mode in DSR. However, we implement the promiscuous learning of source routes from data packets. This follows the specification given in the Internet Draft of DSR. Our reason for not allowing promiscuous listening is that, besides introducing security problems, it cannot be supported in any IP stack where the routing protocol is in the application layer and the MAC protocol uses multiple channels to transmit data. The second difference in our implementation is that since the routing protocol in our stack does not have access to the MAC and link queues, we cannot reschedule packets that have already been scheduled over a link (for either DSR or DST). Tables 3.1 and 3.2 show the constants used in the implementation of DSR and DST, respectively.

3.8.1 Scenarios used in comparison

We compared DSR and DST using three types of scenarios. In all three scenarios, we used the "random waypoint" model described in [7]. In this model, each node begins the simulation by remaining stationary for *pause time* seconds and then selects a random destination and moves to that destination at a speed of 20 m/s for a period of time uniformly distributed between 5 and 11 seconds. Upon reaching the destination, the node pauses again for *pause time* seconds, selects another destination, and proceeds there as previously described, repeating this behavior for the duration of the simulation. We used the speed of 20m/s, which is approximately the speed of a vehicle, because it has been used in simulations in earlier papers [7, 12] and thus provides a basis for comparison with other protocols.

In the first two scenarios, we used a 50 node ad-hoc network, moving over a flat space of dimensions 7 X 6 miles (11.2 X 9.7 km) and initially randomly distributed with a density of approximately one node per square mile. Two nodes can hear each other if the attenuation value of the link between them is such that packets can be exchanged with a probability p, where $p > 0$. Attenuation values are recalculated every time a node moves. Using our attenuation calculations, radios have a range of approximately 4 miles (135 db).

We have random data flows, where each flow is a peer-to-peer constant bit rate (CBR) flow and the data packet size is kept constant at 64 bytes. Data flows were started at times uniformly distributed between 20 and 120 seconds and they go on till the end of the simulation.

We also vary the pause times: 0, 30, 60, 120, 300, 600 and 900 seconds as done in [7].

In the first scenario, there are 20 CBR sources, each of which establishes a connection with a randomly picked destination.

In the second scenario, the number of sources is fixed at 10 sources. There are two runs, one with 40 flows and the other with 60 flows. In the case of 40 flows, each source has peer-to-peer connections with 4 destinations and in the case of 60 flows, each source has peer-to-peer connections with 6 destinations. The total load on the network is kept constant at 32 packets per second in both cases, in order to reduce congestion.

In the third scenario, we attempt to recreate the situation where the ad-hoc network has a single point of attachment to a wired network. We use a 20 node network in a area of 3.15 X 4.25 miles (5 X 6.8 km) where each node is moving with pause time 0. The point of attachment is picked randomly in successive runs of the simulation. In each run, 7 randomly picked wireless nodes have a peer-to-peer flow to and from the point of attachment. The uplinks have a lower rate (0.5 pkts/s) than the downlinks (4 pkts/s) to mimic typical World Wide Web behavior. To add further similarity, the uplink flow is started before the downlink flow.

3.8.2 Metrics used

We used the following metrics in comparing the protocols:

- *Packet delivery ratio*: The ratio between the number of packets received by an application and the number of packets sent out by the corresponding peer

application at the sender.

- *Control Packet Overhead*: The total number of routing packets sent out during the simulation. Each broadcast packet/unicast packet is counted as a single packet.

- *Hop Count*: The number of hops a data packet took from the sender to the receiver.

- *End to End Delay*: The delay a packet suffers from leaving the sender application to arriving at the receiver application. Since dropped packets are not considered, this metric should be taken in context with the metric of packet delivery ratio.

Packet delivery ratio gives us an idea about the effect of routing policy on the throughput that a network can support. It also is a reflection of the correctness of a protocol.

Control packet overhead has an effect on the congestion seen in the network and also helps evaluate the efficiency of a protocol. Low control packet overhead is desirable in low-bandwidth environments and environments where battery power is an issue.

In ad-hoc networks it is sometimes desirable to reduce the transmitting power to prevent collisions. This will result in packets taking a larger number of hops to reach destinations. However, if the power is kept constant, the distribution of the number of hops data packets travel through is a good measure of routing protocol efficiency.

Average end-to-end delay is not an adequate reflection of the delays suffered by data packets. A few data packets with high delays may skew results. Therefore,

we plot the cumulative distribution function of the delays. This plot gives us a clearer understanding of the delays suffered by the bulk of the data packets. Delay also has an effect on the throughput seen by reliable transport protocols like TCP.

3.8.3 Simulation results

Scenario 1

Fig. 3.7 and Fig. 3.8 show the results for scenario 1, which is almost identical to the one presented in [7], barring the differences in the MAC protocols used. There are 20 CBR sources each of which picks a random destination to send traffic to.

Fig. 3.7.a shows the control packet overhead for varying pause times. From the graphs, we see that both DSR and DST have reduced control overhead as the mobility rate drops. DST has about 10% of the overhead of DSR for pause time 0. However, for pause times larger than 120 seconds, DST overhead is about half that of DSR. DST maintains an almost constant overhead. This is due to the fact that the updates in DST contain the entire routing table, which means that nodes running DST have a higher chance of knowing paths to destinations for whom no route discovery has been performed in the past.

As shown in Fig. 3.7.b, the percentage of data packets delivered is almost the same for both protocols. About 80% of the data packets are received for pause time 0. However, the number of data packets received quickly increases to almost 95% at pause time 15. As mentioned earlier, we keep the load on the network constant. Since this load is divided among a large number of flows, we see very little congestion and therefore most packets get through at higher pause times during which the topology

is close to static.

Fig. 3.8.c shows the hop distribution for pause time 0. All the other pause times have similar graphs. The hop distribution results for both protocols are similar, with DSR sending slightly more one hop packets and DST sending more two-hop packets than DSR.

Fig. 3.8.d shows the cumulative delay of both the protocols. The graphs shown are logarithmic in time to accommodate the wide variation. We see that the delay performance of DST is much better than DSR. All of the data packets are sent within 9 seconds for DST. DSR, on the other hand, has packets that have almost 50 seconds of delay.

Scenario 2

Fig. 3.9 and Fig. 3.10 show the results for 40 flows. Fig. 3.11 and Fig. 3.12 show the results for 60 flows. Fig. 3.9.a and Fig. 3.11.a show the amount of control packet overhead each protocol incurs for varying pause times. In both protocols control packet overhead is a function of the workload and the changes in link connectivity. The control overhead of DST is about 34% of the DSR control overhead. Due to the nature of on-demand routing protocols, both protocols show higher overhead when there are flows to more destinations. This is the reason that the control packet overhead for 60 flows is higher than that of 40 flows.

Fig. 3.9.b and Fig. 3.11.b show the percentage of data packets received. This metric shows very similar behavior for both DSR and DST, rising rapidly after pause time 15.

Fig. 3.10.c and Fig. 3.12.c show the hop distribution of the data packets. We show the results only for pause time 0. All the other pause times have similar graphs. The hop distribution shows a shift left for DST in both workloads. This indicates that DST is slightly more efficient in picking shortest path routes. This is because the replies are broadcasted in DST which allows multiple replies to get back to the sender, allowing the sender more choice while picking routes.

Fig. 3.10.d and Fig. 3.12.d, show the delay behavior of the data packets. For the case of 40 flows the graph is inconclusive. DSR shows better performance at lower delays. However, as the delay increases beyond 0.2 seconds DST performs better. In the case of 60 flows DSR performs marginally better.

Scenario 3

In scenario 3 we pick 7 points of attachment randomly. The control packet overhead and the number of data packets received are shown for each point of attachment. The hop count distribution and cumulative delay distribution are averaged over all 7 points of attachment.

From Fig. 3.13.a, we see that DST sends fewer control updates for 6 points of attachment and DSR sends fewer control updates for one. When the first destination tries to set up a connection with the point of attachment, DST sends substantially more replies than DSR. However, when subsequent destinations try to set up a connection with the point of attachment, DST sends far less queries because more nodes know a route to the point of attachment. DST delivers more data packets than DSR for all points-of-attachment, as seen in Fig. 3.13.b.

DST has more of a left shift than DSR in the graph of Fig. 3.14.c, indicating shorter routes. The delay performance of DST is also marginally better than DSR, as shown in Fig. 3.14.d.

3.9 Performance Evaluation in NS-2

We also implemented DST in the latest version of ns-2 in order to compare performances with DSR and AODV in a new environment. We implemented DST in version 9 of ns-2 and used the AODV implementation in the version 9 release. For DSR, we used the version 8 release. For AODV and DSR we used the standard constants released with the ns-2 code. For DST we used the same constants listed in Table.3.2. All protocols used link layer notifications for loss of neighbors. Neighbors were discovered when a packet was received from the neighbor. Since our objective is to test the routing protocols as stand alone protocols we have not considered MAC layer interactions like the promiscuous mode of operation used in DSR. The MAC layer used is IEEE 802.11 distributed co-ordination function (DCF) for wireless LANs, which uses an RTS/CTS/DATA/ACK pattern for unicast packets and DATA for broadcast packets. The physical layer approximates a 2 Mbps DSSS radio interface. The radio range is 250m. Nodal movement occurs according to the random waypoint model with a speed of 20m/s.

3.9.1 Scenarios used

To study performance in light load and heavy load, we set up two scenarios. Both involve 30 node networks in an area of 1000m by 500m and are run for 600

seconds. Each scenario is run for various pause times (0,15,30,60,120,300,600) to study the effect of mobility on the protocols. Each data point consisted of three independent runs based on randomly generated topologies.

In the first scenario we have 10 CBR flows with independent sources and destinations at each point in time. Each flow lasts for 150 seconds and has a rate of 4 packets/sec with packet size of 512 bytes. After one flow ends, we start another flow by picking a random source and destination, thus keeping the load on the network constant. Using this technique, we approximate more closely the working of an ad-hoc networks where flows can start and stop at any time and not necessarily when the network comes up.

In the second scenario we have 20 CBR flows with independent sources and destination at each point in time. Each flow has a rate of 4 packets/sec and therefore the network has a higher load of 80 packet/sec. This scenario will help us test the performance of the protocols when the number of flows in the network is increased.

3.9.2 Metrics used

Two of the metrics we use are the same as in the CPT simulations. They are packet delivery ratio and control packet overhead. Besides these two, we study the average end to end delay suffered by data packets. This is an important metric for delay-sensitive applications like audio and video flows. The final metric we consider is normalized routing overhead. This is defined as the number of bytes of routing overhead per byte of data sent. The routing overhead is calculated at the MAC layer thus giving us an understanding of how much congestion is seen at the MAC layer

due to the routing and how much battery power is used up for routing. The following equation shows the calculation of routing overhead for a broadcast packet and a unicast packet.

$$\begin{aligned} Routing\_overhead(unicast\_packet) &= RTS + CTS + MACheader && (3.1)\\ &\quad + ACK + IPHeader + Packetsize && (3.2)\\ Routing\_overhead(broadcast\_packet) &= MACheader + IPHeader && (3.3)\\ &\quad + Packetsize && (3.4)\\ & && (3.5) \end{aligned}$$

For DSR we did not consider the size of the source rate headers of the data packets in the routing control overhead because of lack of intrumentation.

3.9.3 Simulation results

Scenario 1

In this scenario we test the protocols in an environment of low loads and low number of flows. As shown in Fig.3.15.a, AODV has the least control packet overhead in high mobility conditions and DST has the least overhead in low mobility conditions. This is because of the highly reactive nature of AODV. It sends error packets only along existing routes, whereas DST broadcasts them to all neighbors. AODV also only sends single replies whereas DSR sends multiple replies.

Fig.3.15.b shows the packet delivery ratio. All protocols deliver more packets when the pause time increases and the network becomes more static. AODV does a much better job than DST and DSR because it does not use cached or old information

that causes a loss of packets at the MAC level for both the other protocols.

In the delay graph of Fig.3.16.c, we see that DST has the least delays. AODV is about thrice as much as DST at high mobility and twice at low mobility. This indicates that DST transports packets along very efficient routes because of its more proactive nature.

The normalized routing load graph of Fig.3.16.d indicates that AODV is more efficient in high mobility scenarios and DST is more efficient in low mobility scenarios. This is because DST carries much more routing information in each packet. Therefore, there is lesser need to query as new flows start. DSR is efficient in medium mobility scenarios where stale cached information gets purged due to updates related to mobility.

Overall, we see that AODV is efficient in a high mobility, low load scenario and DST is more efficient in a low mobility scenario. The performance of DSR always lies somewhere in the middle because it of its aggressive caching behavior. Another observation is that performance of all protocols is worse around pause time 60. This is because there is more scope of partitions getting established and more queries being sent as a result. When all nodes are moving constantly as in pause time 0 there is more chance of finding a neighbor for connectivity with the rest of the network.

Scenario 2

In this scenario we see the effect of increasing the number of flows and the load on the behavior of the protocols. In Fig.3.17.a, we see that DST has the least control overhead in all mobility scenarios. The control overhead of DST is half of

AODV in high mobility scenarios and an order of magnitude lesser than AODV in low mobility scenarios. This is because the DST information is already saturated and does not increase with the number of flows.

Fig.3.17.b shows the percentage of data pkts transported. DST send slightly more packets than DSR or AODV because of the routing information maintained.

In the delay graph of Fig.3.18.c, we see that DST has the least delays which again indicates highly efficient routes similar to the case of 10 flows. The reason for the poor delay behavior of AODV is that AODV frequently will not have routes when flows are started and packets sit in the buffers waiting for routes to destinations. In DSR and DST because more replies are sent, there are always paths available and packets are dispatched faster.

We see in the normalized routing graph of Fig.3.18.d that the larger updates of DST do not cost more simply because the extra routing information carried results in fewer packets sent. DST is the most efficient for all mobility scenarios.

3.10 Conclusions

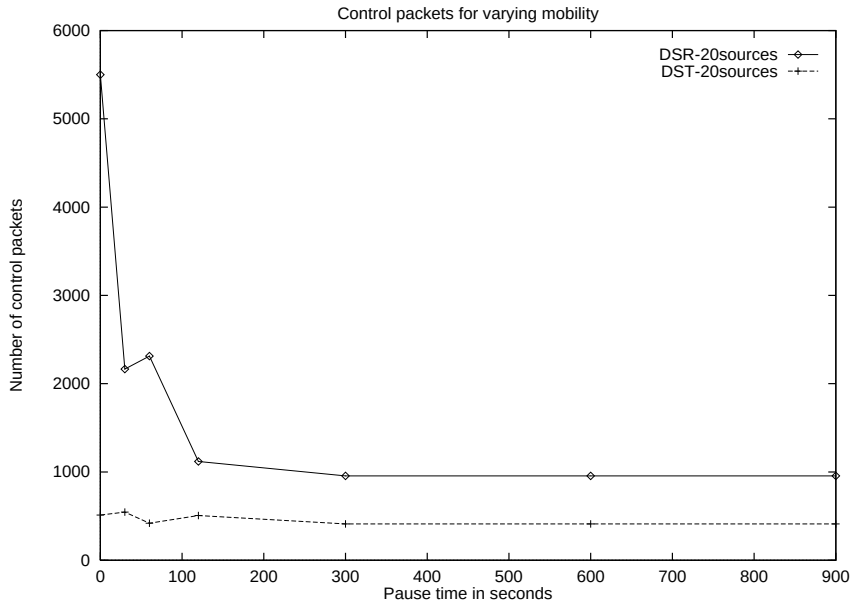
This chapter presented our work on DST. DST does not use sequence numbers and therefore is not prone to inefficiencies in the presence of node failures; this leads us to suggest that our scheme of using only local timestamps to prevent indefinite route queries can be adopted by other one-demand routing protocols to prevent routing inefficiencies. DST also does not use reliable updates or polling of neighbors. This implies that DST creates substantially less overhead than protocols that use the above

features. We introduce conditions that reduce control packet overhead at the expense of non-optimal routes, all the while preventing permanent looping of data packets. The proof of correctness shows that in the presence of reliable updates, the routing algorithm would create loop-free routes and the protocol would converge in finite time.

Extensive simulations were used to compare our protocol against DSR and AODV, which are existing efficient on-demand routing protocols. Our simulation in CPT show that in most high mobility scenarios, DST shows an order of magnitude less control overhead than DSR. For other scenarios, it performs consistently better than DSR with respect to control overhead. The results for delay, hop count and percentage of data packets received are mixed. In some cases DSR performs better and in others DST performs better, which leads us to believe that both protocols are at par when performance in these metrics is taken into consideration.

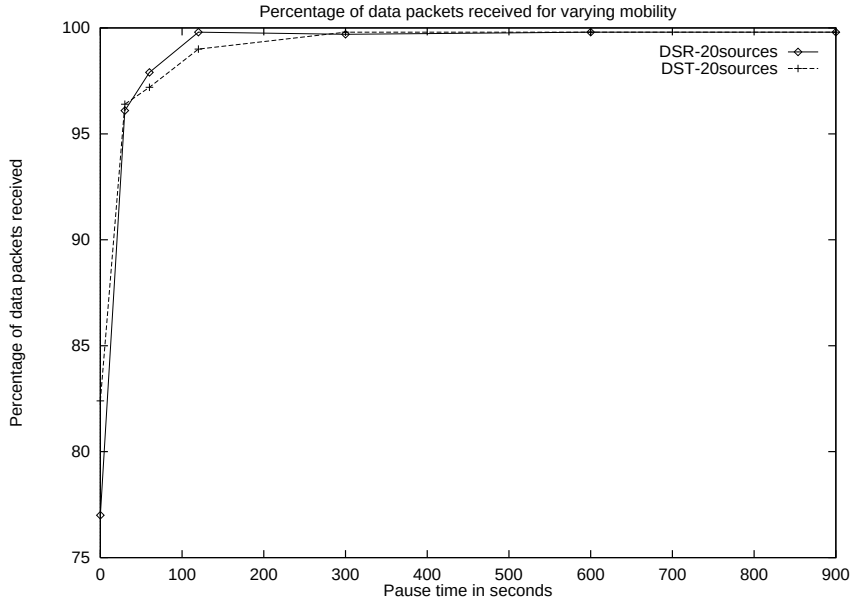
The ns-2 simulations show that DST is very efficient protocol for high flow scenarios regardless of mobility. AODV is a more efficient protocol in high mobility combined with low load scenarios. Regardless of the number of flows and mobility, DST has the best delay behavior. This leads us to suggest that DST is an ideal protocol for delay sensitive applications like audio flows.

Finally, all our simulations results show that DST is very suitable for ad-hoc networks and incurs limited control overhead.



(a)

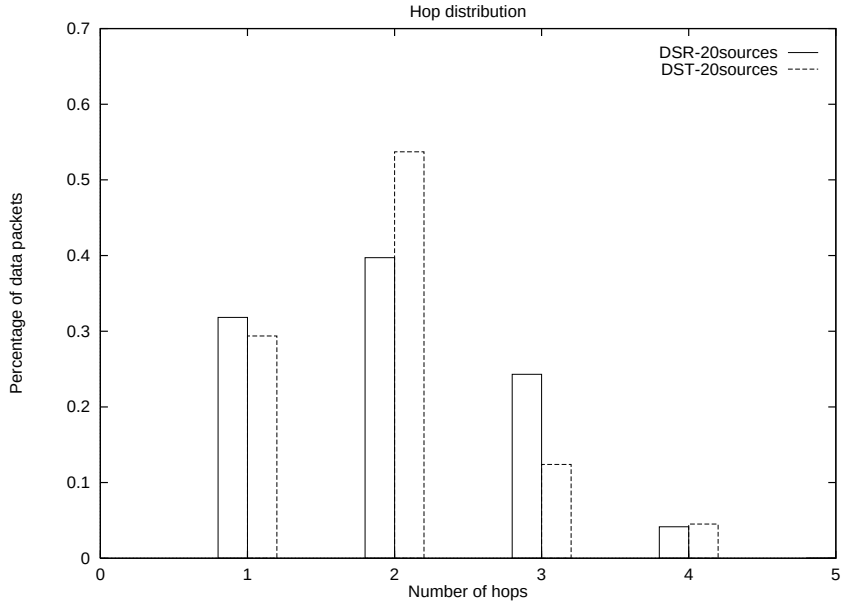
Number of control packets sent



(b)

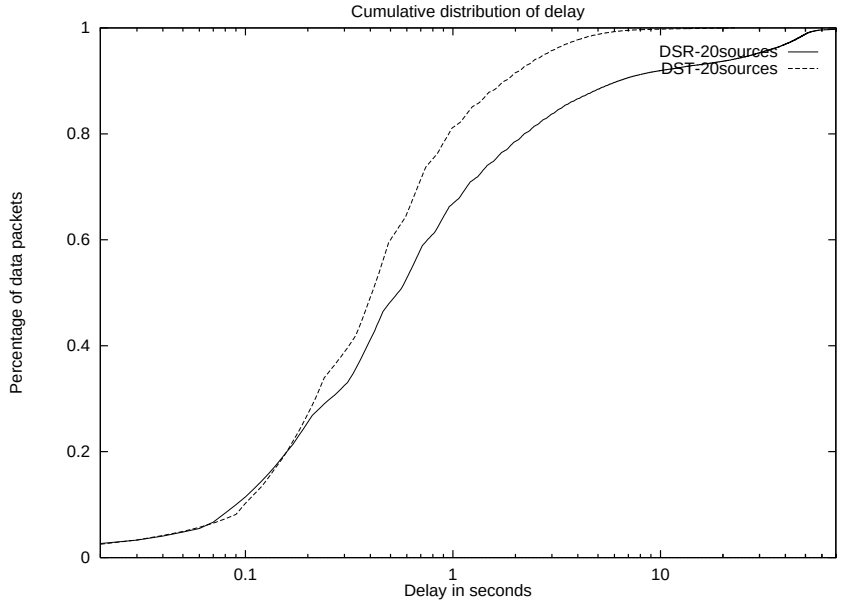
Percentage of data packets received

Figure 3.7: Results for 20 sources picking random destinations for peer-to-peer flow



(c)

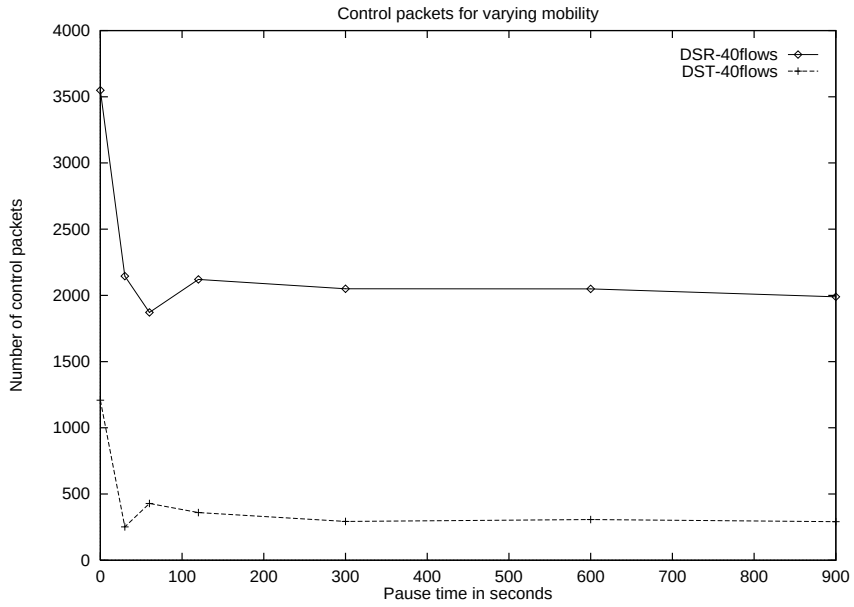
Hop count distribution (*pause time 0*)



(d)

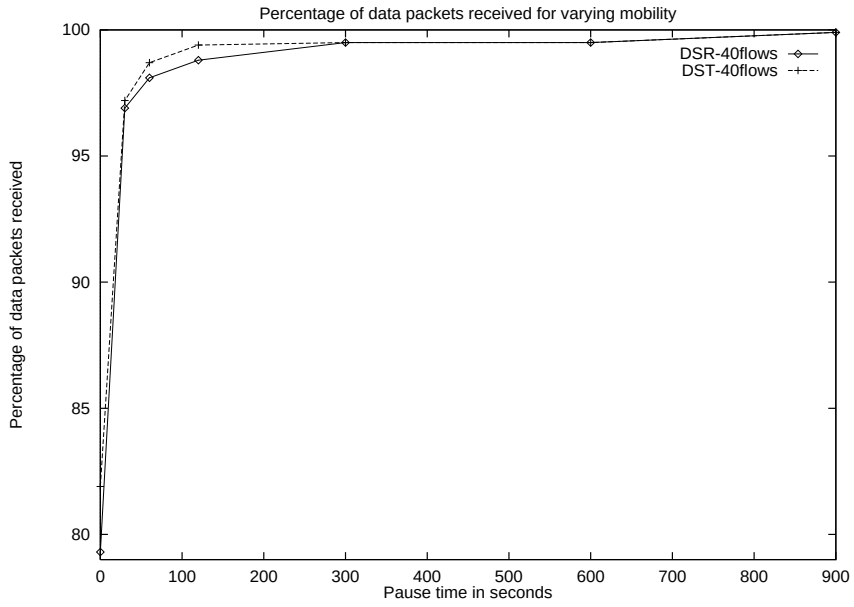
Delay cumulative distribution (*pause time 0*)

Figure 3.8: Results for 20 sources picking random destinations for peer-to-peer flow



(a)

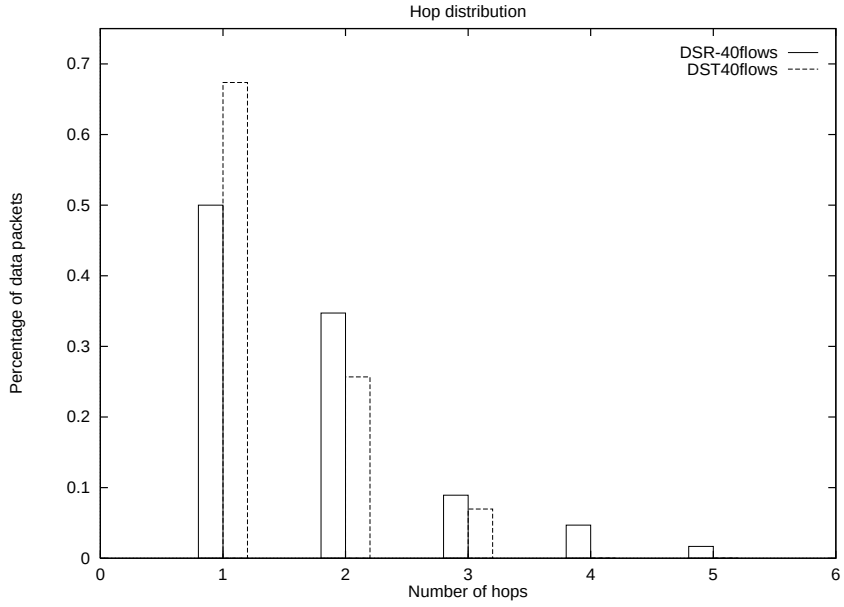
Number of control packets sent



(b)

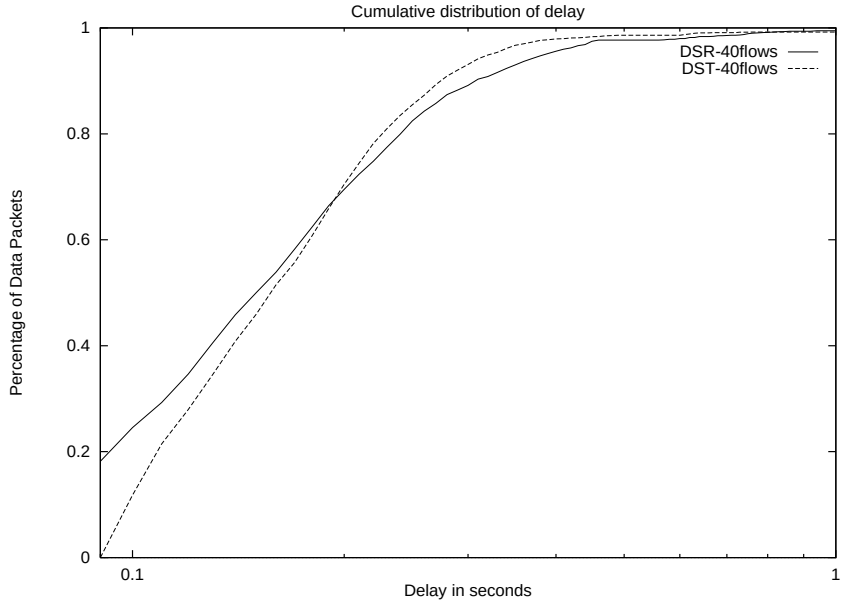
Percentage of data packets received

Figure 3.9: Results for 40 flows - 10 sources with 4 destinations each



(c)

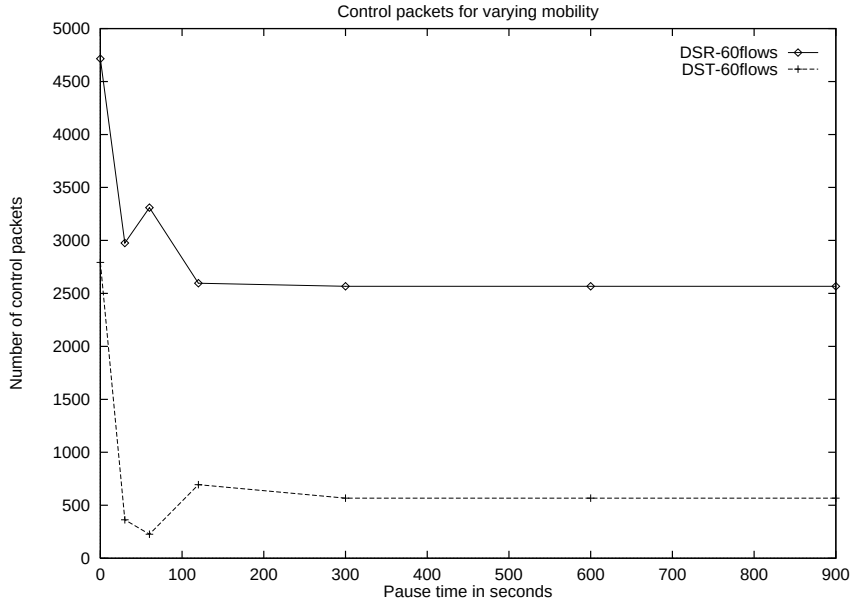
Hop count distribution (*pause time 0*)



(d)

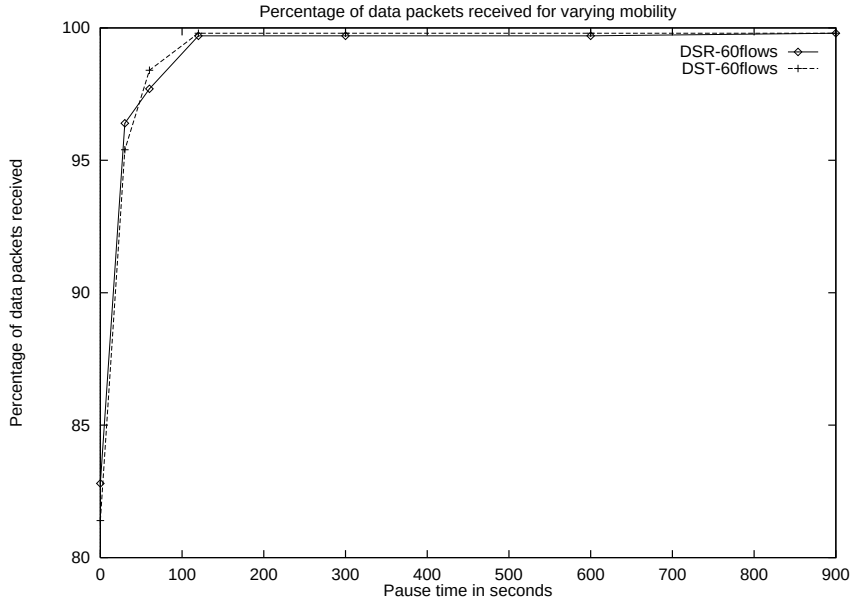
Delay cumulative distribution (*pause time 0*)

Figure 3.10: Results for 40 flows - 10 sources with 4 destinations each



(a)

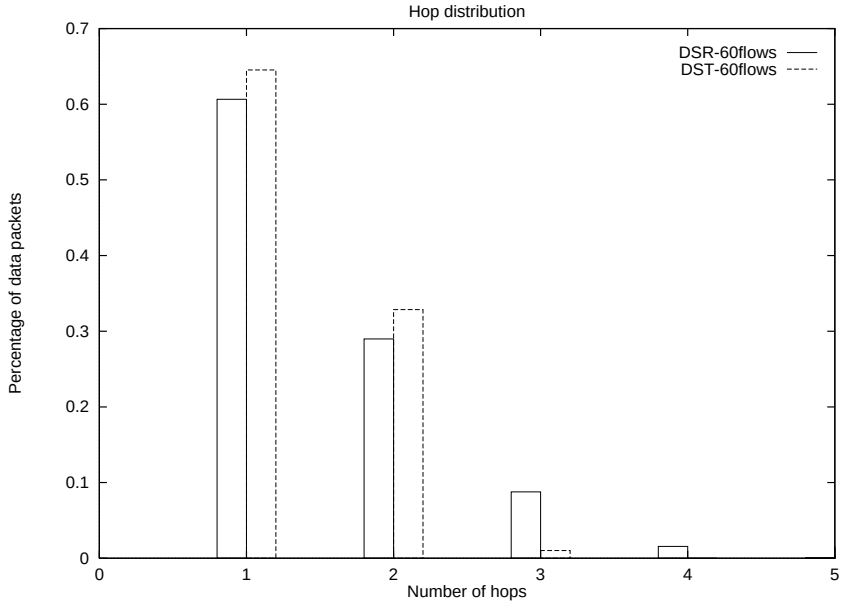
Number of control packets sent



(b)

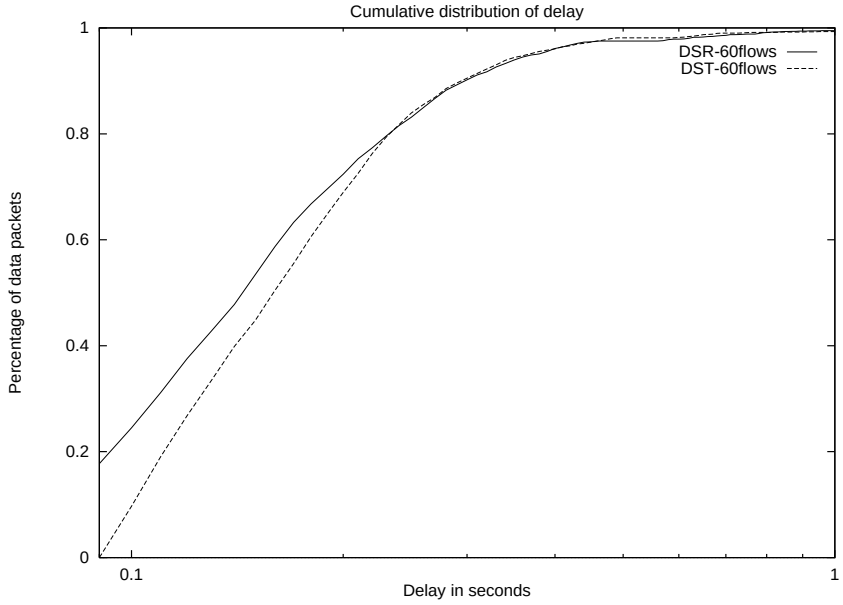
Percentage of data packets received

Figure 3.11: Results for 60 flows - 10 sources with 6 destinations each



(c)

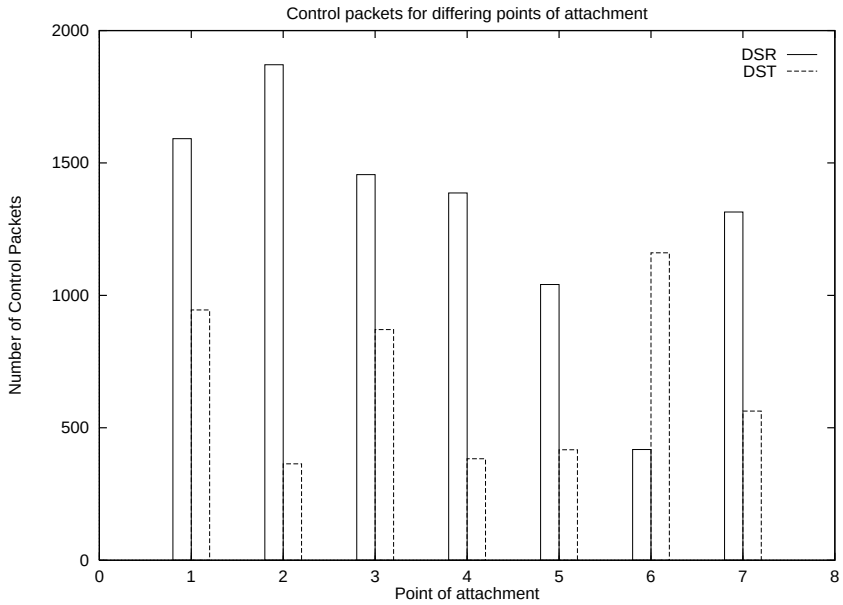
Hop count distribution (*pause time 0*)



(d)

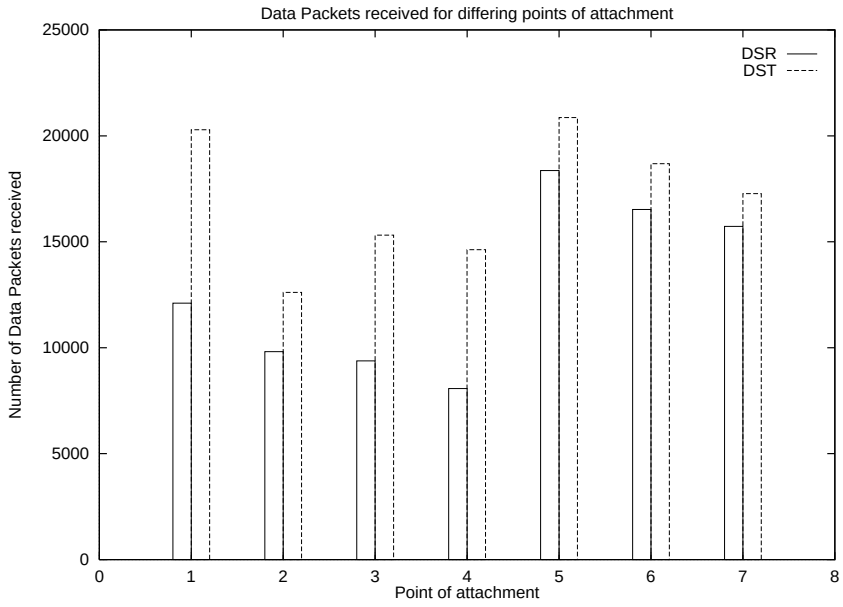
Delay cumulative distribution (*pause time 0*)

Figure 3.12: Results for 60 flows - 10 sources with 6 destinations each



(a)

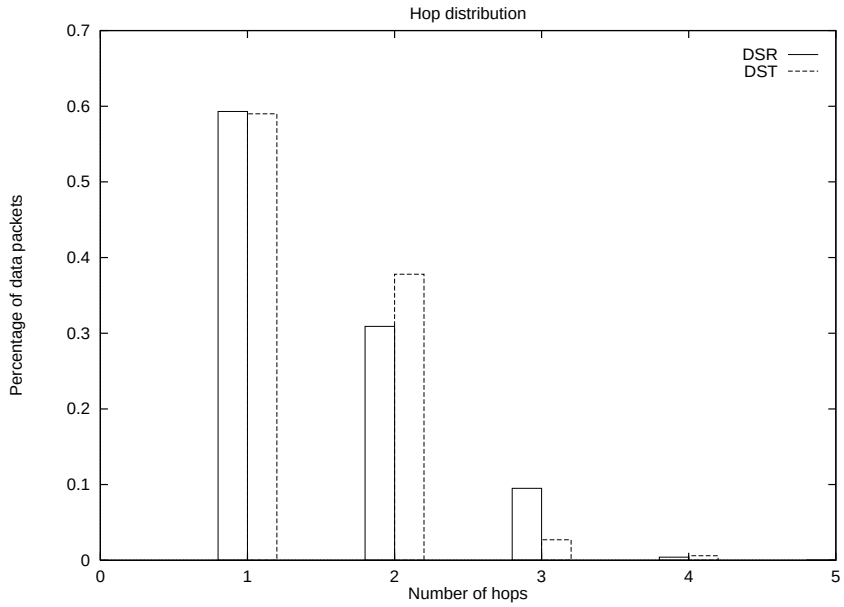
Number of control packets sent



(b)

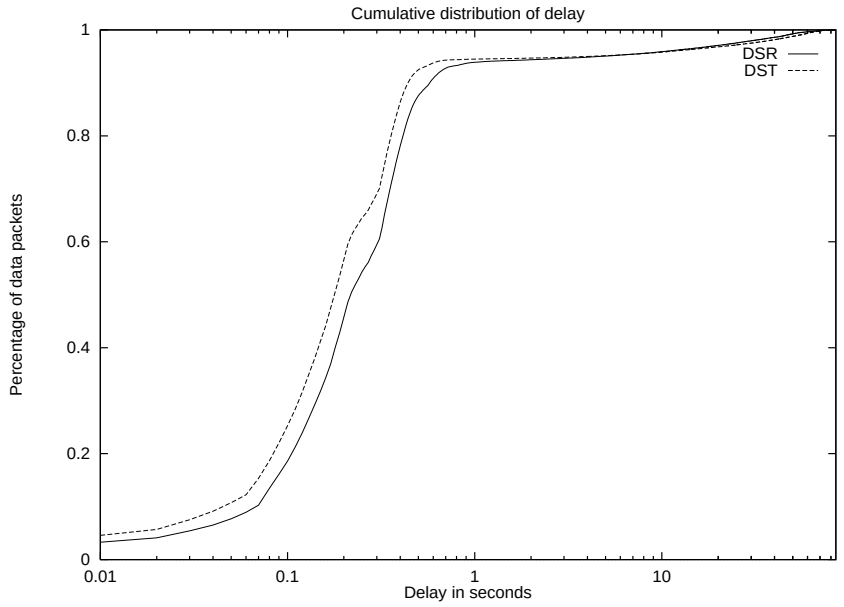
Number of data packets received

Figure 3.13: Results for single point of attachment scenario



(c)

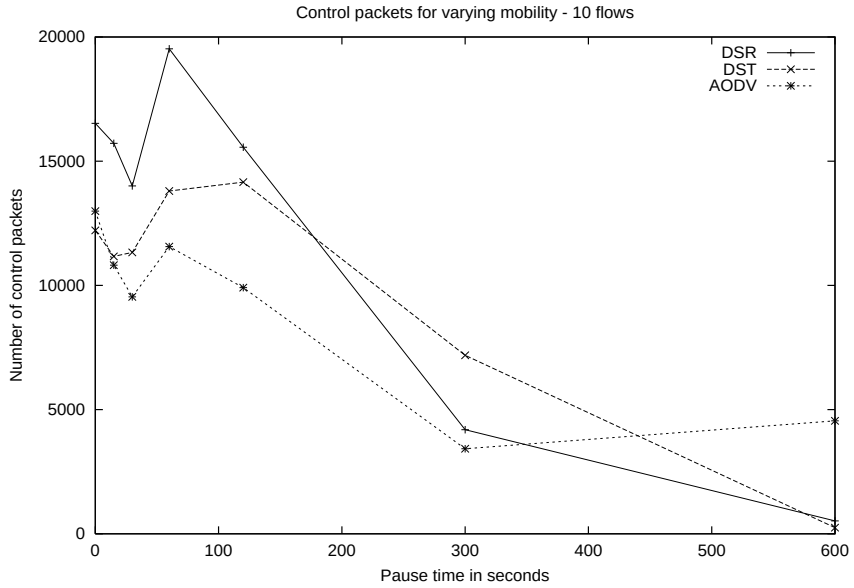
Hop count distribution averaged over all points of attachement



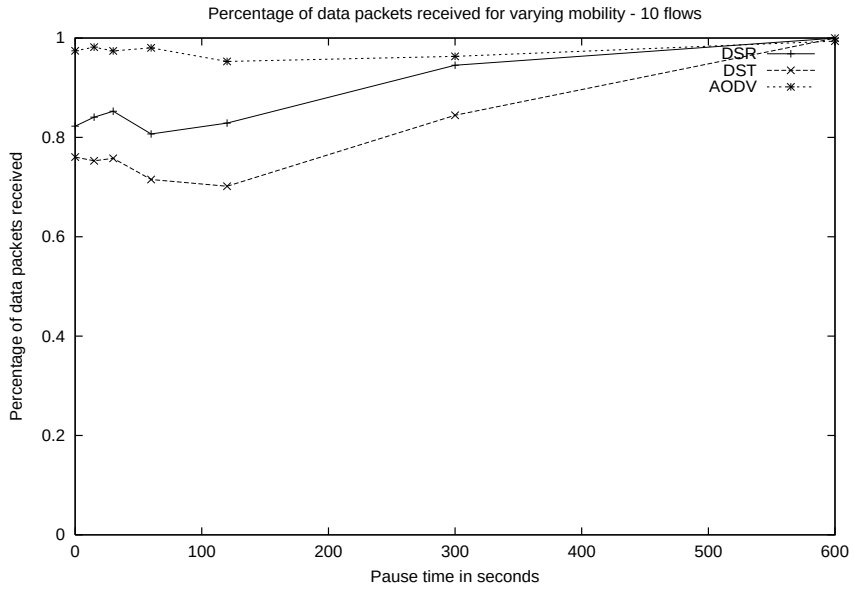
(d)

Cumulative distribution function of delay averaged over all points of attachment

Figure 3.14: Results for single point of attachment scenario

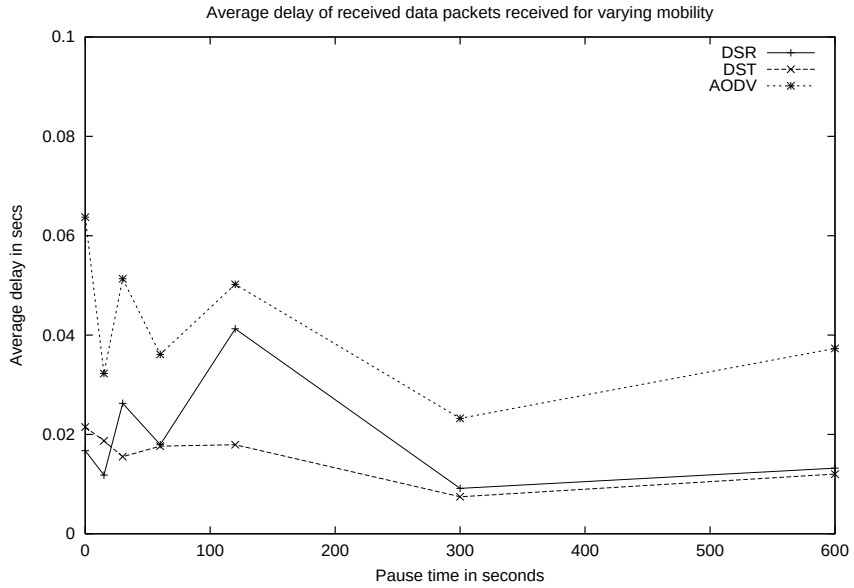


(a)
Number of control packets sent

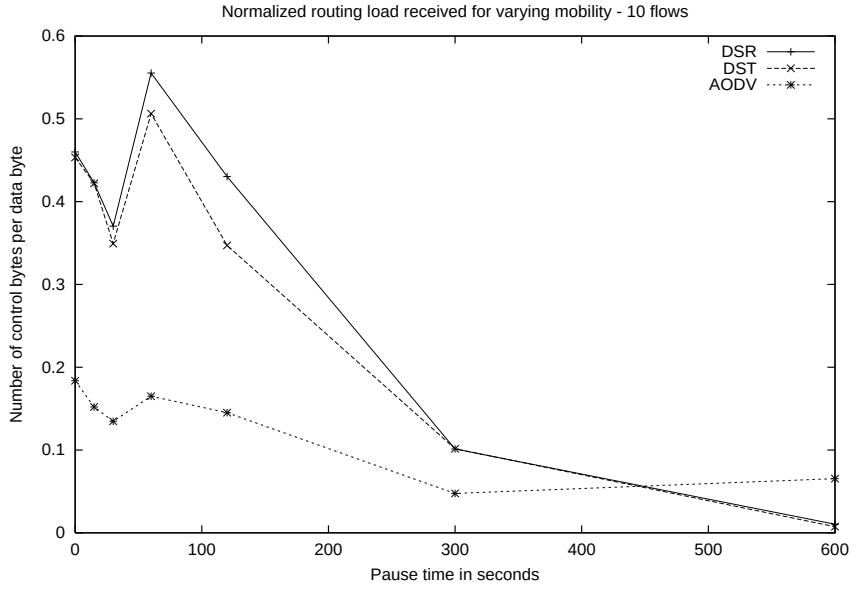


(b)
Percentage of data packets received

Figure 3.15: Results for 10 flows in a 30 node network (ns2)

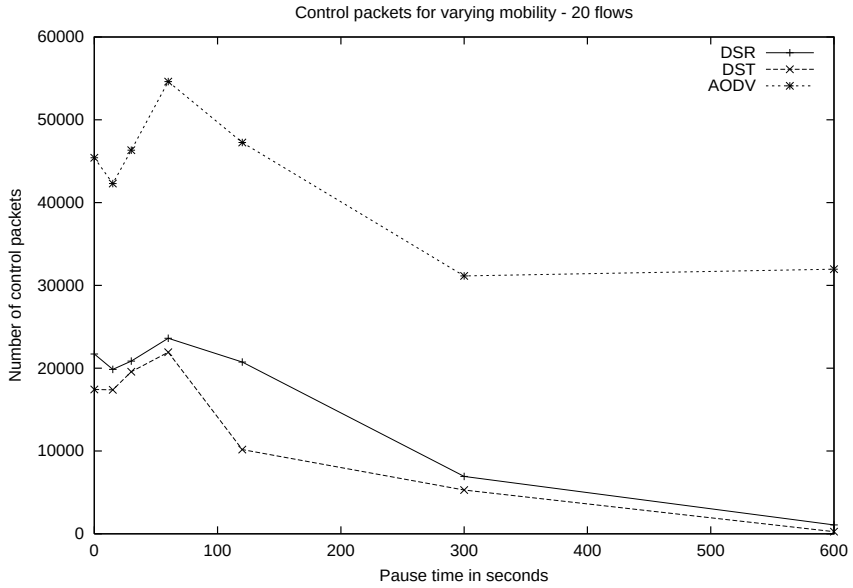


(c)
Average end to end delay

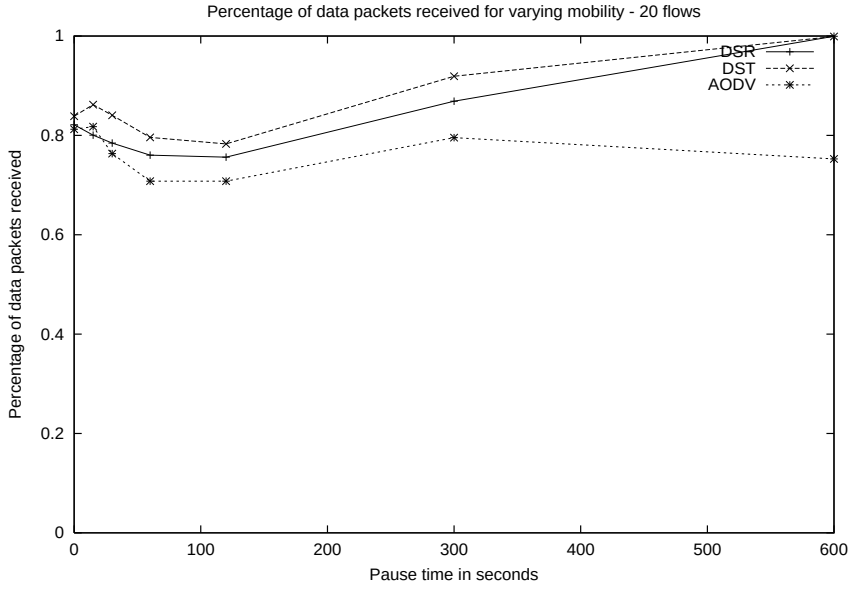


(d)
Normalized Routing Load

Figure 3.16: Results for 10 flows in a 30 node network (ns2)

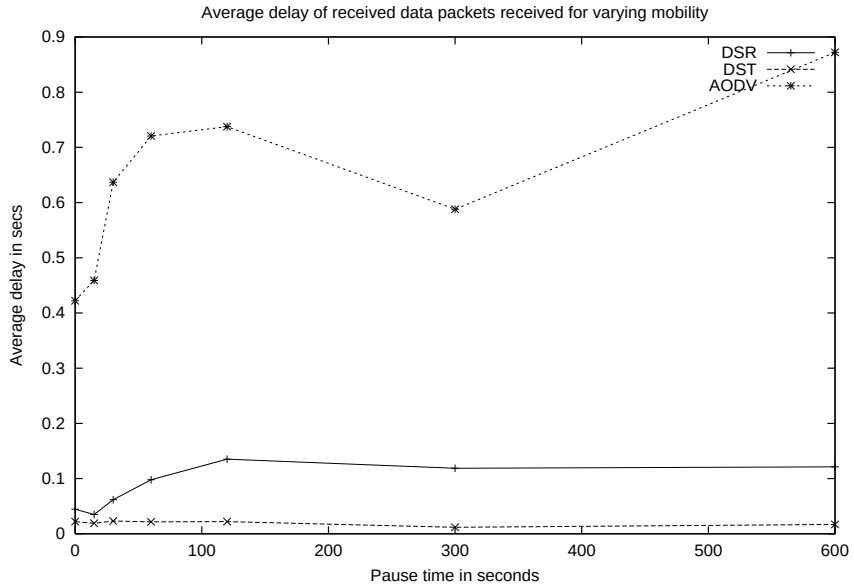


(a)
Number of control packets sent

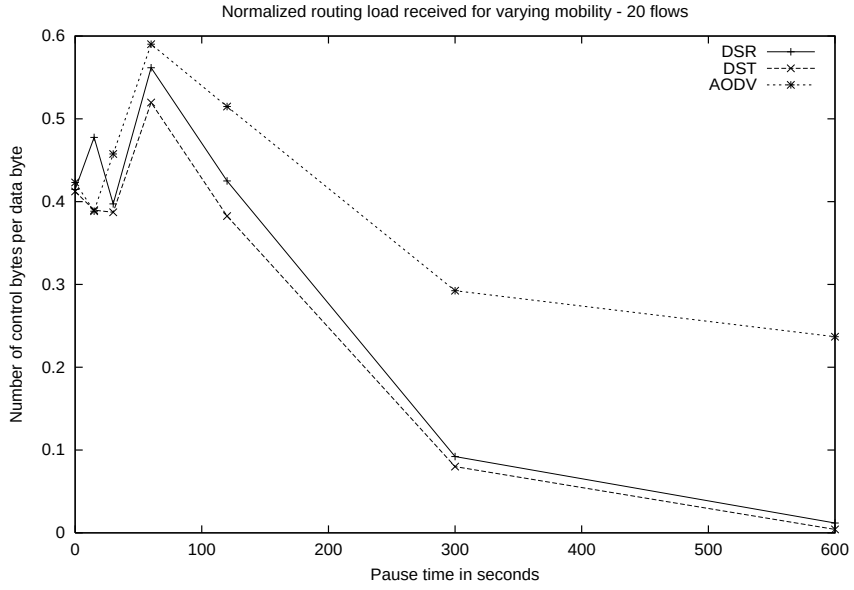


(b)
Percentage of data packets received

Figure 3.17: Results for 20 flows in a 30 node network (ns2)



(c)
Average end to end delay



(d)
Normalized Routing Load

Figure 3.18: Results for 20 flows in a 30 node network (ns2)

Chapter 4

Bandwidth Efficient Table Driven Source Tracing

Several recent studies have been published comparing the performance of the wireless routing protocols using different simulators, mobility models and performance metrics [7, 16, 12] . Based on their results, all of these papers conclude that on-demand routing protocols perform better than table-driven routing protocols. However, all the table-driven routing protocols tested use the optimum routing approach. In other words, these protocols try to maintain shortest paths at all times. A consequence of maintaining shortest paths is that if the topology of the network changes rapidly, the control overhead increases dramatically.

In this piece of work, our aim is to study the effect of relaxing the requirement for shortest paths in a table driven routing protocol and determining if this can lead to solutions whose performance is equivalent to on-demand routing approaches. Our goal is to design a table-driven distance-vector routing protocol that uses the same constraints used in on-demand routing protocols, i.e. paths are used as long as they are valid and updates are only sent when a path becomes invalid. The new protocol

BEST (Bandwidth Efficient Source Tracing) [42, 45], uses source tracing for non-optimum table-driven routing. The reason why prior table-driven routing protocols have been unable to perform non-optimum routing is that these protocols have used either distances to destinations or topology maps to predict paths to destinations. None of these techniques allow a router to discern if the path picked by it conflicts with its neighbors, resulting in "counting to infinity" problems. Consequently, these protocols have to send updates in order to avoid loops, and the best that can be done is that the updates are sent periodically. However, in BEST, the paths used by neighbors are maintained and this allows the design of a distance-vector protocol with non-optimum routing and event-driven updates, resulting in reduced control overhead. BEST also does not assume reliable updates which allows us to use the same network model used in DST.

4.1 Routing Structures maintained

The routing structures maintained in BEST are a subset of those maintained by DST, i.e, a *routing table* and a *distance table*. BEST also does not maintain any packet buffer for data packets waiting for routes. As in any table driven routing protocol, packets are sent if there is a valid route and they are dropped if there is no valid path at the moment of arrival.

The routing table at router i contains entries for all known destinations. Each entry consists of the destination identifier j, the successor to that destination s_j^i, the second-to-last-hop to the destination p_j^i, the distance to the destination D_j^i and a route

tag tag_j^i. When the element tag_j^i is set to *correct*, it implies a loop-free finite value route. When it is set to *null*, it implies that the route still has to be checked and when it is set to *error*, an infinite metric route or a route with a loop is implied.

The distance table at router i is a matrix containing, for each known neighbor k and each destination j, the distance value of the route from i to j through k, D_{jk}^i and the second-to-last hop p_{jk}^i on that route. D_{jk}^i is always set equal to $RD_j^k + l_k^i$, where RD_j^k is the distance reported by k to j in the last routing message and l_k^i is the link cost of link (i, k). The link cost may be set to one reflecting hop count or it may be set to some other link parameter like latency, bandwidth, etc.

4.2 Routing information exchanged

Routing update messages are broadcast to all neighbors. Each packet contains the address of the sender and a list of routing table entries, where each entry specifies a destination, the distance to the destination and the predecessor to the destination. Having a reliable MAC layer would allow incremental updates. However, the protocol as presented here accounts for unreliable routing updates.

All data packets contain the source and the destination and are unicast reliably by the link layer.

4.3 Routing Table Updating

Routing tables are updated under two conditions, the first condition being the receipt of an update message and the second condition being a detection of a link status change.

4.3.1 Receiving an update

The processing of an update in BEST is done in the same manner as in DST. When an update from neighbor k is received, the entries in the distance table corresponding to neighbor k are updated. The paths to each destination are then recomputed. BEST sends updates only if any of the following conditions have been met.

1. A node discovers a new destination with a finite and valid path to the destination.

2. A node loses the last path to a destination.

3. A node suffers a distance increase to a destination.

4. A node changes the next hop to a destination.

From the above conditions, it follows that no update is sent if the distance to a destination decreases. However, an update is sent when the distance to a destination increases or a node changes the next hop, because this condition has the potential to cause a loop.

Two more conditions are added to prevent permanent looping due to unreliable broadcasts.

5. A node sends a unicast update to a neighbor that sends it a data packet, if the neighbor is in the path from it towards the destination.

6. A node sends a unicast update to a neighbor that sends it a data packet, when the path implied by the neighbor's distance table entry is different from the path

implied by the node's routing table.

In both these conditions, the data packets are dropped. Permanent looping can occur when nodes are not aware of the latest changes in their neighbor's routing tables. The use of conditions 5 and 6 can be explained with the help of an example shown in Fig. 4.1.a. The node addresses are marked in bold font. Node j is the required destination. The path to j implied by traversing predecessors from j is marked in italics. Initially, all nodes have loop-free routes. The loss of links (i, j) and (m, j) and the loss of update packets from i and m can result in a loop shown in Fig. 4.1.b. When i gets a data packet from k, it finds that its distance table entry for k implies the path ij, while i's own path implies $ilmj$ which is different from ij. Therefore due to condition 6, the data packet is dropped and a unicast routing update is sent resulting in k setting its path to kmj. Now, when k gets a data packet from m, it sends a unicast update to m because m is its successor on the path to j. This follows from condition 5. When m gets the update, it detects a loop and resets its distance to infinity, thus breaking the loop.

4.3.2 Topology/Link-Cost Changes

When the link layer protocol can no longer send a data packet to a neighbor, an indication is sent to the routing layer, in response to which the link to the neighbor is marked with value infinity, and all the distances are recomputed. If the path to any destination is lost, then an update is sent.

When the routing protocol gets a link up signal from the link layer protocol, it broadcasts an update and includes the neighbor k in the distance table with all

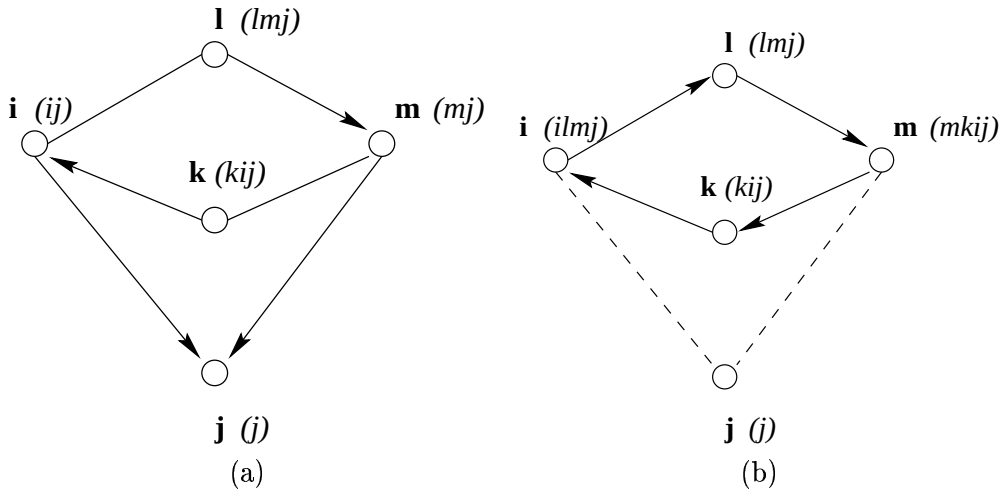


Figure 4.1: Creation of a permanent loop due to unreliable updates

distances through k set to infinity. One exception is the distance of k through k, which is set to zero.

4.4 Proof of Correctness

In this section, we show that the basic routing algorithm used in BEST is correct for the routes present in the routing table. The aim is to prove that all routes are loop-free.

The following assumptions are made about the behavior of the routers and the network.

- The link layer can inform the routing protocol within a finite time after a link fails, a link comes up or a link cost changes.
- Control packets are exchanged reliably.

We use the following notation to aid the proof

l_j^i: Link cost for link (i, j).

D_j^v: Distance in the routing table entry for destination j at router v.

D_{jk}^v: Distance in the distance table entry for neighbor k for destination j. This can also be defined as the distance from router v to destination j through neighbor k.

s_j^v: Successor (next-hop) in the routing table for destination j which is picked according to the path selection rules of BEST.

p_j^v: Predecessor (second to last hop) in the routing table entry for destination j at router v.

p_{jk}^v: Predecessor in the distance table entry for neighbor k for destination j.

Definition 1

The link cost l_j^i for link (i, j) can be extracted from the distance table at router v if there is a column in the distance table for a neighbor k such that $l_j^i = D_{jk}^v - D_{ik}^v$ and $p_{jk}^v = i$. Similarly, the link weight can be extracted from the routing table as $D_j^v - D_i^v$, where $p_j^v = i$.

Lemma 1

If a routing table is generated based on the rules of BEST, any link cost that can be extracted from this routing table can be extracted from a column of the distance table.

Proof

Let N_v denote the neighbors of router v. Let l_j^i be a link cost extracted from the routing table of node v. One of the required conditions for updating distance D_j^v, successor $s_j^v = k$ and predecessor p_j^v is that the distance table entry for k indicates

the shortest distance to all nodes in the implied path to j. Therefore, we know that if $s_j^v = k$, then $D_j^v = D_{jk}^v$ and $D_i^v = D_{ik}^v$, which implies that the link cost can be extracted from the distance table. Therefore, Lemma 1 is true. □

Lemma 2

The link cost change of a link will be reflected within a finite time in the routing and distance tables of a neighboring router.

Proof

An link can either go down, suffer a link cost change or come up. If an link goes down, then the link layer protocol will inform the routing protocol within finite time because the data packets will not get through on that link. Similarly, if the link layer protocol is monitoring neighbors, then the link layer protocol will inform the routing layer of the link cost change or a link coming up. In response to a loss of neighbor the routing protocol will delete the column entry corresponding to the neighbor from the distance table. In response to a link cost change, the link cost is marked in the distance table. In both cases, the change in link cost triggers a reevaluation of the routing table. In response to the link coming up a complete update is sent to the neighbor. When an update is received, the distance and routing table are updated. Therefore, this lemma is true. □

Theorem 1

If the distance entries in the distance table and routing table are finite, then a path

can be extracted from the distance and routing table entries and this extracted path is loop-free.

Proof

Fig. 4.2 depicts a permanent loop caused when node a shifts its successor from node g

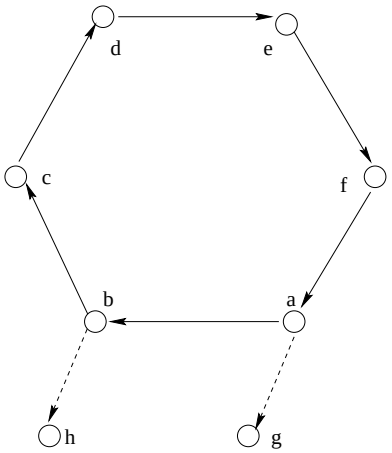


Figure 4.2: Figure depicting a permanent loop

to node b due to a distance change at node a at time T. The head of the arrow in the path is the successor of the tail of the arrow. Without loss of generalization we assume that the route is for a certain destination j. Assume at some time before T node b had switched successors from h to c. There are two cases that we need to consider. In the first case, node b had suffered a distance increase and therefore informed its neighbors, including a of the new path it was using. Since there was no loop before time T, we can make the assumption that $b, c, d, e, f, a, g...j$ was a simple path. In the second case, node b picked node c because link (b, c) decreased in cost thus making distance through node c shorter than distance through h. Since node b changed a successor, node b would have to inform node a of its new simple path $b, c, d, e, f, a, g...j$. At time

T when node a needs to look for a new path, it steps through the path implied if it is used b. This path would be $a, b, c, d, e, f, a, g...j$, which would trigger the condition that node should not accept a path that includes itself as an intermediate node, thus preventing a loop. Therefore, theorem 1 is true. □

Theorem 2

Following a link cost change, there can only be a finite number of updates generated for that change.

Proof

Suppose the link cost change happens at time T. If the link cost decreases, then no update is sent because there is no distance increase for any destination. Therefore, for this case the theorem is true.

Now we consider the cases where the link cost increases, link goes down or comes up. Using lemma 2, we know that the change is recorded in a node adjacent to the link. The change of this link causes either a distance increase, or the change of a successor to a destination or the announcement of a new destination. This causes the node adjacent to the link to send out a single update. A node that receives the update will send out further updates only if the update arrives from the successor to a destination, else the node just processes the update and changes its distance table and sends no more updates. If the update arrives from a successor, then a further update is sent under the following conditions

1. The update increases the distance to a destination.

2. The update announces a new destination.

3. The update causes a change of the successor.

Without loss of generalization we consider updates for a single destination. In Theorem 1, we proved that the finite paths extracted from the routing table entries are loop-free. Let us consider such a path to destination j at node v. Consider the case where node v sends an infinite number of updates. This is only possible because its present successor k in the path is sending it an infinite number of distance increase updates or updates that cause change of successor. Node k can send infinite updates only if its successor in turn is sending infinite updates. If we step through the loop free path to j, we finally end up at node j, which cannot be sending infinite updates for itself. Therefore, by contradiction, this theorem, is true. □

Theorem 3

Within a finite time after the failure of a link (i,k) in the network at time T, all routers using link (i,k) will stop assuming link (i,k) exists.

Proof

If node i is not using link (i, k) to reach any destination j at time T, then none of the nodes using i as a forwarding node will be using link (i, k). Therefore, no updates need to be sent or processed after the event. Suppose node i is using link (i, k) and it fails. This will cause a distance increase at node i because link (i, k) was part of the shortest path of node i to destination j. Hence, node i will send out an update with the new path not containing link (i, k). This update will cause further updates until

it reaches a node where both of the cases are true:

- The link (i, k) is not part of the path implied in the routing table.
- The update from the neighbor neither causes a distance increase nor causes the node to switch a successor.

Because the link layer can detect a link failure within a finite time and a node creates and processes an update within a finite time and all paths containing (i, k) are finite loop-free paths, all nodes will stop assuming link (i, k) exists within a finite time. □

Theorem 4

If a node does not have a path to destination j at time T, then the distance between the two routers is set to infinity for all time after the loss of the link that caused the loss of path.

Proof

It follows from Theorem 3 that the loss of the link will be communicated to all routers that use the link in a path to a destination within a finite time after T. From algorithmic decription, we know that the loss of a finite path, causes the distance to be set to infinity. □

4.5 Performance Evaluation

We ran simulations for two different experimental scenarios to compare the average performance of BEST against the performance of DSR and DST. All three protocols are implemented in *CPT*, which is a C++ based toolkit that provides a

wireless protocol stack and extensive features for accurately simulating the physical aspects of a wireless multi-hop network. The protocol stack in the simulator can be transferred with a minimal amount of changes to a real embedded wireless router. The stack uses IP as the network protocol. The routing protocols directly use UDP to transfer packets. The link layer implements the IEEE 802.11 standard [3] and the physical layer is based on a direct sequence spread spectrum radio with a link bandwidth of 1 Mbit/sec.

To run DSR in CPT, we ported the DSR code available in the *ns*2 [17] wireless release. There are two differences in our DSR implementation as compared to the implementation used in [7]. Firstly, we do not use the *promiscuous* listening mode in DSR. We, however, implement the promiscuous learning of source routes from data packets. This follows the specification given in the Internet Draft of DSR. Our reason for not allowing promiscuous listening is that, besides introducing security problems, it cannot be supported in any IP stack where the routing protocol is in the application layer and the MAC protocol uses multiple channels to transmit data. The second difference in our implementation is that since the routing protocol in our stack does not have access to the MAC and link queues, we cannot reschedule packets that have already been scheduled over a link (for either DSR, DST or BEST). Tables 3.1 and 3.2 in the previous chapter show the constants used in the implementation of DSR and DST, respectively.

4.5.1 Scenarios used in comparison

We compared DSR, DST and BEST using two types of scenarios. In both scenarios, we used the "random waypoint" model described in [7]. In this model, each node begins the simulation by remaining stationary for *pause time* seconds and then selects a random destination and moves to that destination at a speed of 20 m/s. Upon reaching the destination, the node pauses again for *pause time* seconds, selects another destination, and proceeds there as previously described, repeating this behavior for the duration of the simulation. We used the speed of 20m/s (72 km/hr), which is the speed of a vehicle, because it has been used in simulations in earlier papers [7, 12] and thus provides a basis for comparison with other protocols. All simulations are run for 900 seconds. In both scenarios, we used a 50 node ad-hoc network, moving over a flat space of dimensions 7 X 6 miles (11.2 X 9.7 km) and initially randomly distributed with a density of approximately one node per square mile.

Two nodes can hear each other if the attenuation value of the link between them is such that packets can be exchanged with a probability p, where $p > 0$. Attenuation values are recalculated every time a node moves. Using our attenuation calculations, radios have a range of approximately 4 miles (135 db).

Scenario 1 mimics the behavior of an emergency network or a network set up for military purposes. Scenario 1 is almost identical to to the one presented in [7], barring any differences due to implementation of the MAC protocols. We have 20 random data flows, where each flow is a peer-to-peer constant bit rate (CBR) flow with a randomly picked destination and the data packet size is kept constant at 64

bytes. Data flows were started at times uniformly distributed between 20 and 120 seconds and they go on till the end of the simulation at 900 seconds. We did 7 runs of the simulation where each run had different sets of source-destination pairs. The total load on the network is kept constant at 80 data packets per second (40.96 kbps) to reduce congestion. Our rationale for doing this is that increasing the packet rate of each data flow does not test the routing protocol. On the other hand, having flows with varying destinations does so. We also vary the pause times: 0, 30, 60, 120, 300, 600 and 900 seconds as done in [7].

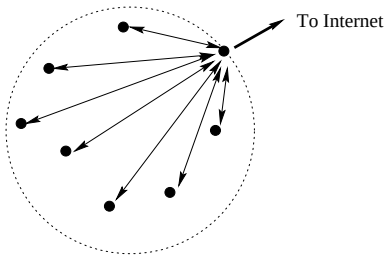


Figure 4.3: Scenario 2

Scenario 2 mimics the applications of ad-hoc networks as wireless extensions to the Internet. In this case, one or two nodes act as points of attachment of the ad-hoc network to the Internet. Accordingly, all Internet traffic travels to and from the attachment points as shown in Fig. 4.3. To model this situation, we pick one node as the point-of-attachment to the Internet for a simulation run of 900 seconds and we do five such runs and plot our results. During each run, the sender node first establishes a low rate connection (5.85 kbps) with the point-of-attachment. Immediately after the forward connection is established, the backward connection is started from the point-of-attachment to the sender. This connection has a higher rate of 40.96 kbps. Each

pair of connections lasts for 300 seconds. In each epoch of 300 seconds, we start seven pairs at random times. This setup closely resembles number of nodes accessing the Web through the point-of-attachment. We run our simulations for two pause times, 0 (continuous movement) and 900 (no movement).

4.5.2 Metrics used

In comparing the protocols, we used the following metrics:

- *Packet delivery ratio*: The ratio between the number of packets received by an application and the number of packets sent out by the corresponding peer application at the sender.
- *Control Packet Overhead*: The total number of routing packets sent out during the simulation. Each broadcast packet/unicast packet is counted as a single packet.
- *Hop Count*: The number of hops a data packet took from the sender to the receiver.
- *End to End Delay*: The delay a packet suffers from leaving the sender application to arriving at the receiver application. Since dropped packets are not considered, this metric should be taken in context with the metric of packet delivery ratio.

4.5.3 Performance results

Scenario 1

Fig. 4.4.a shows the control packet overhead for varying pause times. An obvious result is that the control packet overhead for all the three protocols reduces

as the pause time increases. BEST and DST are about 34 % better than DSR at pause time zero. At low rates of movement, DST is a clear winner with one third the control packet overhead of BEST and one tenth the control packet overhead of DSR. Clearly, the fact that the updates in DST contain the entire routing table, means that nodes running DST have a higher chance of knowing paths to destinations for whom no route discovery has been performed in the past. We are able to mimic the behavior of table-driven routing protocols in low topology change scenarios, in that we almost have information about the entire topology with very few flood searches.

As shown in Fig. 4.4.b, the percentage of data packets delivered is almost the same for DST and BEST. At lower pause times, DSR has the same packet delivery ratio as DST and BEST. However, as the pause time decreases, DSR suffers due to data packets getting dropped at the link layer, indicating that the routes provided in the source routes are not correct any more. At lower pause times, links get broken faster. Even though this results in higher control overhead, the routes obtained are relatively new. As mentioned earlier, we keep the load on the network constant. Since this load is divided among a large number of flows, we see very little congestion and therefore most packets get through at higher pause times during which the topology is close to static.

For Fig. 4.5.c we collated the hop count values for data packets during all pause times and plotted the hop distribution. All three protocols have almost the same number of one hop packets, indicating that the zero hop query is very effective in obtaining routes to neighbors. However, for the number of hops greater than one, we

see that BEST performs the best. This is expected of a table driven routing protocol that tries to maintain valid routes at most times. DST's behavior is slightly worse than BEST. DSR on the other hand sends packets through longer routes. This is a direct consequence of the fact that after the initial query-reply process DSR pretty much uses the route it caches, without trying to better them.

Fig. 4.5.d shows the cumulative delay of all the protocols. The graphs shown are logarithmic in time to accommodate the wide variation. We see that BEST performs better than DSR or DST, with DST being very close. Almost all packets are sent within 4 seconds in BEST and within 8 seconds in DST. Some packets in DSR take almost 30 seconds. This is because a packet is allowed to stay in a buffer for a maximum of 30 seconds before it is dropped. These are packets that found the path just in time.

Scenario 2

Fig. 4.6.a and Fig. 4.6.b show the results for the case of continuous movement. We see that BEST has almost double the control packet overhead of DST or DSR. The protocol is essentially reacting to the high rate of topology changes. The traffic does not seem to influence the behavior of BEST, because the same information needs to be maintained no matter what point-of-attachment is used. DSR and DST have almost the same behavior in terms of control overhead. DSR performs well in this traffic pattern, because with every flood search towards the point-of-attachment, the point-of-attachment learns the reverse path to the source from the source route accumulated in the queries. Another reason is that the fast changing topology forces out stale routes

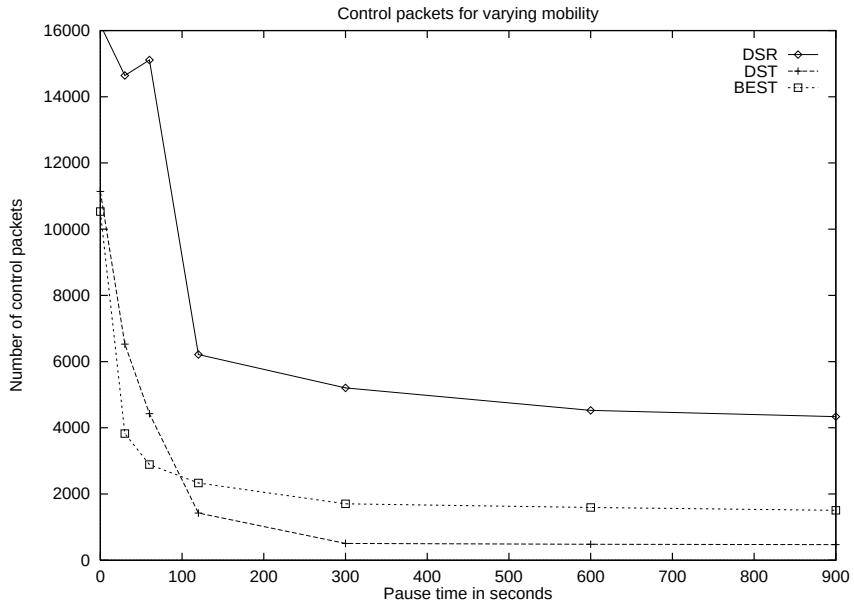
from DSR caches. This also results in DSR sending about 10 % more data packets than DST or BEST as shown in Fig 4.6.b.

Fig. 4.7.c and Fig. 4.7.d show us the results for the static case. This scenario is important because it resembles a static community network, e.g., households with wireless routers used to reach the Internet through an access point. In this case, BEST incurs about 3 times more control overhead than DST, whereas DSR incurs 14 times more control overhead than DST. DST performs this well because the entire network knows the path to the point of attachment with a single flood search. Since there are no topology changes, there is no need for another flood search. BEST also performs much better for a static network than for a dynamic one. No topology changes mean no table driven updates after the initial updates sent when the network comes up. The surprising result is the really bad behavior by DSR, most of which seems to be driven by increase in flood searches caused by old routes. A similar behavior is seen in terms of the ratio of data packets received. DST and BEST lose very few packets, while DSR seems to lose about 50% of them. As congestion due to control packets increases, we observe more and more data packets being dropped.

4.6 Conclusions

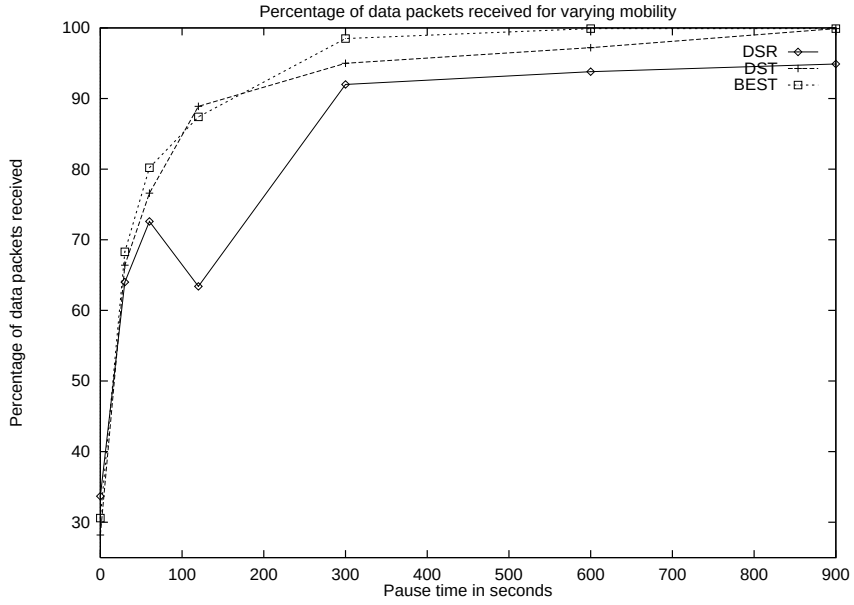
Simple rules were introduced in BEST for the efficient use of source tracing within the context of table-driven routing. The rules used in BEST are simpler than those introduced for STAR [22], which is the only other table-driven routing protocol that has been shown to be as efficient as on-demand routing protocols.

Simulations were used to compare DST, BEST and DSR. The results showed that DST provides comparable average delays and packet delivery ratios while incurring far less control overhead than DSR or BEST. In our first scenario, which closely resembled an ad-hoc scenario for a battlefield or an emergency situation, DST had about one-tenth the control overhead of DSR while delivering packets with the same efficiency as BEST, which is table-driven. BEST, has about one-third the control overhead of DST while having the best results for hop count and delay. For the second scenario, which is comparable to community networks accessing the Internet via wireless links, DSR had almost 14 times more overhead than DST, which suggests that DST is an ideal solution for static community networks. In static networks, the poor performance of DSR in terms of delay and throughput suggests that it needs a mechanism to flush out stale routes in static scenarios. In scenario 2, BEST has double the overhead of DSR and DST when all the nodes are moving. This suggests that a table-driven routing protocol is a wrong choice for scenarios with high topology change and few destinations. On the other hand, BEST delivers almost all the packets and has one fourth the control overhead of DSR for the static version of scenario 2, which implies that it may be used as a solution for community networks, though DST is a better option.



(a)

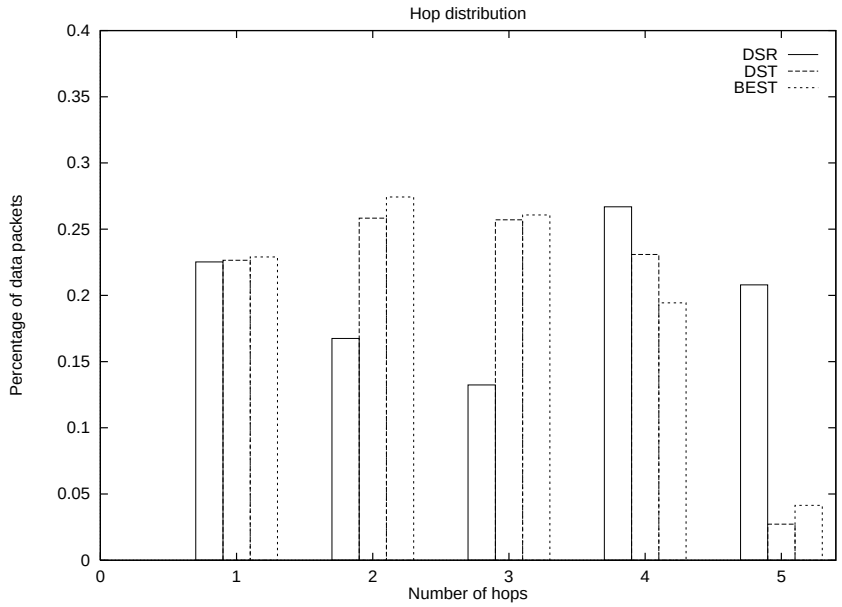
Number of control packets sent



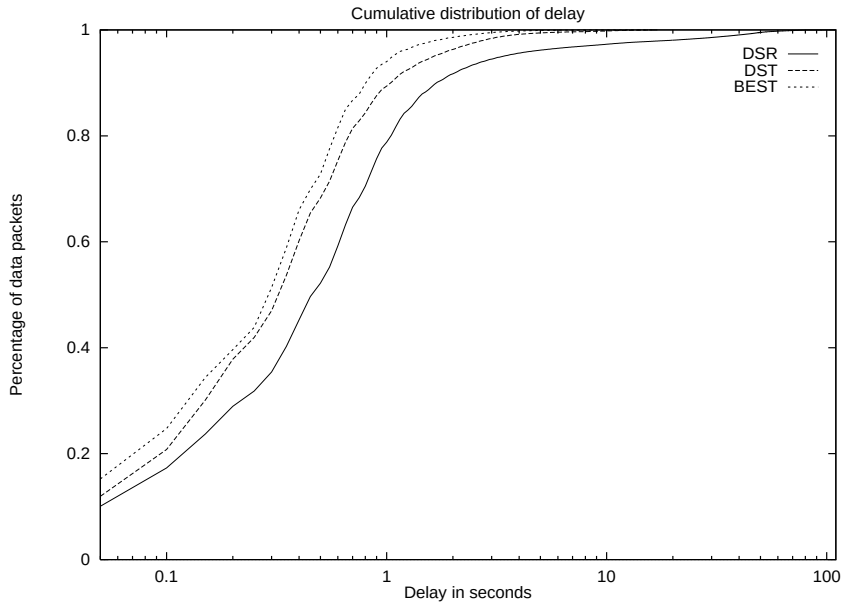
(b)

Percentage of data packets received

Figure 4.4: Results for 20 sources picking random destinations for peer-to-peer flow

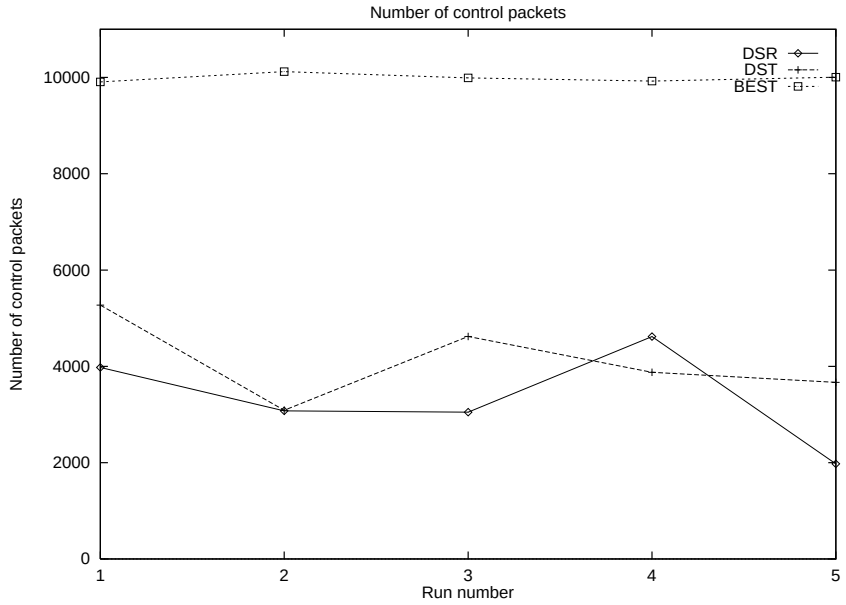


(c)
Hop count distribution



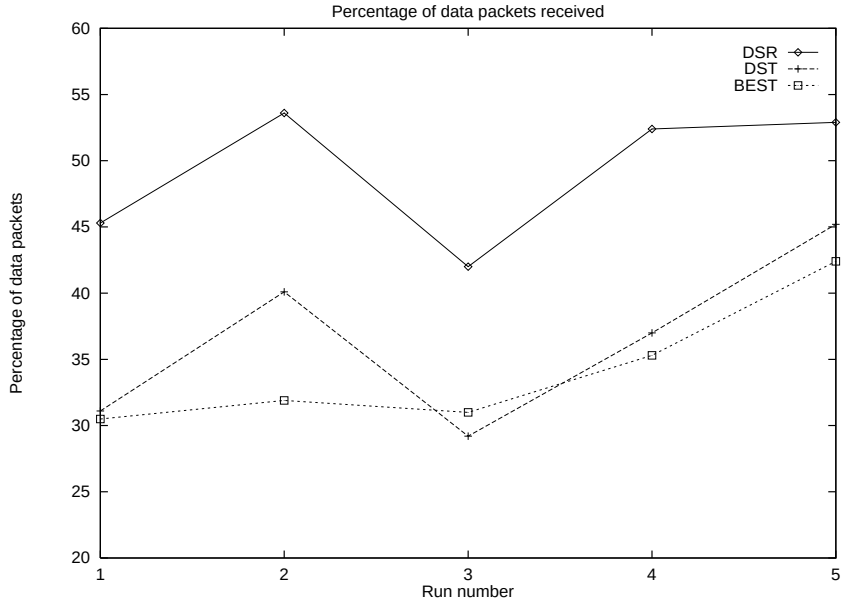
(d)
Cumulative delay distribution

Figure 4.5: Results for 20 sources picking random destinations for peer-to-peer flow



(a)

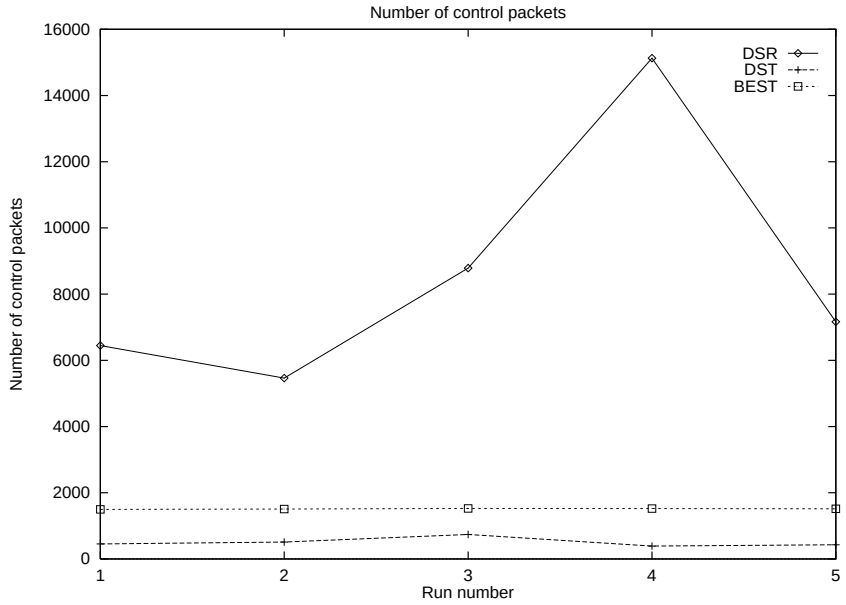
Number of control packets sent



(b)

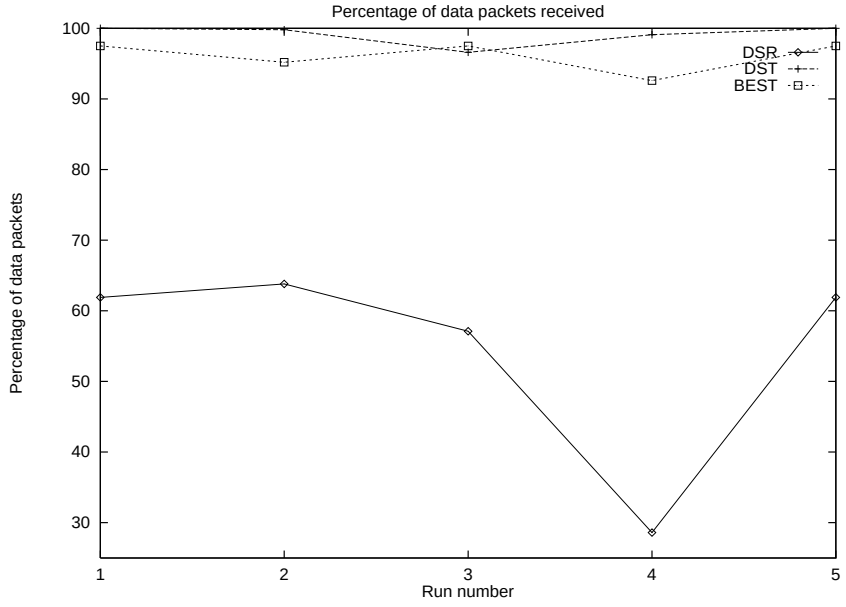
Percentage of data packets received

Figure 4.6: Results for single point of attachment (all nodes moving)



(c)

Number of control packets sent



(d)

Percentage of data packets received

Figure 4.7: Results for single point of attachment (static topology)

Chapter 5

Multipath Routing

The emergence of interactive multimedia communications on the Internet has spurred the development of network services to improve quality of service (QoS) and even attempt to offer some QoS guarantees. In this context, multipath routing has important advantages over single shortest path routing. Multipath routing is able to better exploit the underlying physical network resources, provide greater end-to-end throughput, increase reliability of packet delivery and reduce route oscillation and congestion.

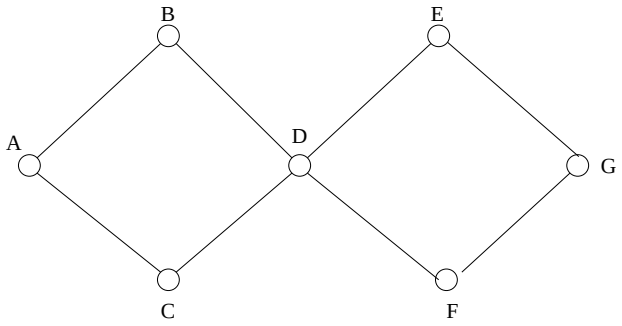


Figure 5.1: Illustration of node disjoint and link disjoint paths

Multipath routing and its applications have been well studied in the networking literature, in particular for wired networks. An early work by Maxemchuk [33] on

an application of multipath routing known as *dispersity routing* discusses how a message can be dispersed among multiple paths by splitting it in order to achieve smaller average delay and delay variance. Since then, there has been a significant amount of work done on multipath routing for both connection oriented [6] and connectionless technologies [37, 52, 51]. In most of the above papers, the emphasis has been on minimizing delay by using multiple paths. Another aspect of multipath routing is path disjointedness. Path disjointedness allows for paths to fail independently. There are two types of disjoint paths: *node disjoint* and *link disjoint*. Node disjoint paths do not have any nodes in common, except the source and the destination. In Fig. 5.1 ABD and ACD are node disjoint paths. Link disjoint paths do not have a common link but may have common nodes. In the figure, $ABDEG$ and $ACDFG$ are link disjoint paths. There has been a body of work on finding node and link disjoint paths in wired networks.

Ogier and Shacham [15] proposed a distributed algorithm that finds shortest pairs of node- (link-) disjoint paths. An improved technique was proposed by Sidhu et al. [5] to compute node-disjoint paths. A well known example of a multipath routing algorithm is OSPF [34] which computes multiple paths of equal cost. More recently, multipath distance vector algorithms that use the concept of diffusing computations to construct and maintain DAGs have been proposed [52]. However, all these algorithms have been designed for wired networks where control packet overhead is not a concern.

Some ad-hoc network routing protocols have built in capability to compute multiple paths. DSR uses source routing, by virtue of which it can obtain multiple

loop-free paths. However, aggressive use of route caching and cache pollution can lead to problems like stale caches and reply storms, which can get accentuated by storing multiple paths. TORA and ROAM can both maintain multiple paths. However, the requirement of reliable, in-order delivery of packets causes high overhead.

Path disjointedness has been considered in ad-hoc routing but mainly in the context of source routing. Nasipuri et al. [4] propose extensions to DSR to compute multiple paths to increase average time between route discovery floods. They study the effect of multiple paths and path lengths on time between route discoveries using analytical modeling. Lee's work on split multipath routing [31] introduces a modified flooding algorithm for multipath DSR in order to increase the chances of obtaining disjoint routes. Pearlman et al. [10] analyze the impact of alternate path routing for load balancing. Marina and Das [32] introduce AOMDV which is a multipath version of AODV that uses destination based sequence numbers to maintain multipaths.

In this work, we introduce multipath DST or MDST which is a multipath on-demand routing protocol that does not need source routes, timestamps or sequence numbers in order to maintain multiple paths correctly. Our work with DST shows that the query reply process in DST lends itself to multipath routing with minor changes and almost no change in packet overhead. We also require no changes in the routing information maintained because DST already stores the routing tables of all its neighbors. One of the key advantages of the DST query reply process is that the path taken by replies is not dependent on the path taken by queries. Therefore, we do not need to modify the flooding algorithm as was done in the work by Lee [31]. DST

replies get propagated in a manner similar to table driven routing, which results in a lot more routes to choose from.

An ad-hoc network can be implemented over a single channel or multiple channels. In a single channel system transmission and reception is done on the same channel. A transmitter first gains control of the channel. When a transmitter is sending data to a receiver, all neighbors will receive signal on the same channel and hence will not be able to receive data from other sources at the same time. In order to support concurrent data transmission, multiple channels need to be employed. In particular, one has to assign a unique channel (frequency, time slot, code) to every receiver or transmitter in a two hop neighborhood [21, 1].

Consider the wireless network in Fig.5.1 where there are two link disjoint paths $ABDEG$ and $ACDFG$. First, we consider the case of a single channel wireless network. The node D that is common to both paths $ABDEG$ and $ACDFG$. While it is relaying data along $ABDEG$, it cannot receive or transmit data along $ACDFG$ because each node is equipped with a single half-duplex transceiver. Furthermore, while D is sending data to E or F, nodes B and C cannot receive from node A because that would result in collisions at node B and C. Next, we consider the case of a multiple channel wireless network. Even in this case, when node D is relaying data along $ABDEG$ it cannot receive or transmit data long $ACDFG$. A multiple channel network, however, does not suffer from collisions of the second kind at node B and C while node D is transmitting. Thus, in the presence of common nodes along the path, traffic on one route will block traffic along other routes, thus providing very minimal throughput

performance enhancement.

However, in the case of a on-demand protocol, maintaining multiple link disjoint paths can reduce the number of route discoveries, because if one route fails, one can switch to the other route as long as the route failure is not due to the common node moving away or the common node failing.

Since our aim with MDST is to study the decrease in end-to-end delays and increase in goodput, we acquire and maintain node disjoint paths in MDST.

5.1 Link and MAC layer assumptions in MDST

As in DST, we assume that each node has a distinct node identifier instead of having a interface identifier. In the ad-hoc network each node has connectivity with all its neighbors using a single physical radio link. This physical broadcast link is modelled as individual point-to-point links between a node and its neighbors. Each link has a positive cost associated with it, and if a link fails its cost is set to infinity. If there is a neighbor protocol that monitors link costs, then those costs are used as link metrics in the routing, else a single hop is considered a metric of one. A node failure is modelled as all the links incident on the node getting set to infinity.

Routing messages are broadcast unreliably and the protocol assumes that routing packets may be lost due to changes in link connectivity, fading or jamming. However, it is assumed that a lower-level protocol can inform the routing protocol when data packets cannot be unicast to the next hop. For the purpose of routing table updating, a node A considers another node B as its neighbor if A receives an

update with distance zero to B. Node B is no longer node A's neighbor when node A cannot send data packets to it.

5.2 Routing information maintained in MDST

Routing information maintained in MDST is almost identical to that in DST. The *routing table*, *data buffer* and a *query table* have the same specifications and variables as used in DST. One additional field is added to the distance table. This is variable tag^i_{jk} which is set in the distance table entry for every neighbor k for every destination j. When the element tag^i_{jk} is set to *correct*, it implies a loop-free finite value route to j through k. When it is set to *null*, it implies that the route still has to be checked and when it is set to *error*, an infinite metric route or a route with a potential loop is implied.

5.3 Routing information exchanged in MDST

There are two types of control packets in MDST - *queries* and *updates*. All control packets headers have the source of the packet ($pkt.src$), the destination of the packet ($pkt.dst$), the number of hops ($pkt.hops$) and an identifier $pkt.type$ that can be set to $QUERY$ or $UPDATE$. Each packet has a list of routing entries, where each entry specifies a destination j, a distance to the destination RD^i_j and a predecessor to the destination rp^i_j.

If the MAC protocol allows for reliable updates then the information sent in each packet can be reduced by only sending incremental routing updates to update routing tables. In this study, however we assume a MAC protocol based on collision

avoidance because that would provide an easy basis for comparison with previous results in routing for ad hoc networks. Therefore, in the rest of this paper, we assume that routers transmit their entire routing tables when they send control messages. Control packet size may affect the delay experienced by packets in the MAC layer but this will be proportional to the number of control packets sent.

Data Packets in MDST only need to have the source and destination in the header.

5.4 Computing multiple node disjoint paths using source tracing

To describe the process of multiple path formation in MDST, we model a network as an undirected graph $G(V, E)$. V is the set of routers in the network and E is the set of links in the network. Let N_i be the neighbors of a node i and let S_j^i denote the set of next-hop choices at node i for forwarding packets to node j. Then the goal of MDST is to maintain a routing graph denoted by the link set $SG_j = \{(m, n) | n \in S_j^m, m \in V\}$ in the presence of link cost changes and link failures, such that it is a directed acyclic graph and such that each $k \in S_j^i$ offers a node disjoint path to the destination.

In DST each node maintains copies of shortest paths of the neighbors to destinations in the distance table data structure. Potentially each of these paths defines an alternate path to the destination. However, accepting the path provided by each neighbor without conditions will result in loops.

We use the term *implied path* to mean the path obtained by stepping through

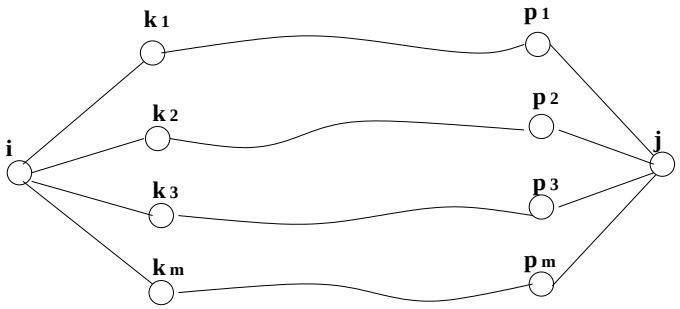


Figure 5.2: Example to illustrate conditions for multiple paths

predecessors to a destination. For instance, in Fig.5.2, one can trace the implied path through neighbor k_1 by starting from the predecessor entry p_1 for the destination j reported by k_1. Next we look, for the predecessor entry reported by k_1 for destination p_1 and so on, resulting in the implied path $[i, k_1, ..., p_1, j]$.

In order to eliminate any possibility of permanent loops, we need to use the following conditions while picking up nodes as successors or neighbors that can join the successor set. The routing table still maintains the shortest path because only the shortest path is exchanged with neighbors. To update the routing table entry for destination j we pick a neighbor k as a successor s_j^i to destination j if both the following conditions are true.

1. k offers the shortest distance to all nodes in the path from j to i.

2. the path from j to k does not contain i and does not contain any repeated nodes.

The distance through this successor is the distance stored in the routing table and exchanged with neighbors. The successor is added to the successor set. To put other neighbors in the successor set S_j^i, we use the following conditions:

1. a neighbor k has to offer the shortest path to all the nodes in its implied path excluding j.

2. the path should not include i and should have no node repeated.

3. the predecessor reported by k should not be the same as the predecessor reported by any other neighbor in the successor set.

The last condition ensures node-disjointedness and we can see why from the following theorem.

Theorem 1: *Having a different predecessor is a necessary and sufficient condition for node-disjointedness of two paths reported by different neighbors.*

Proof: We prove this property by contradiction. Assume that two paths are node disjoint and have the same predecessor. This is a simple contradiction because the link between the predecessor and the destination is common if the predecessor is same. Therefore, two paths that are node disjoint do not have the same predecessor.

Now consider the assumption that at an arbitrary node i, two paths to destination j have different predecessor and are not node disjoint. If the two paths are not node disjoint, then they share the common intermediate node m. This common intermediate node has only one shortest path to destination j and only one predecessor p_j^m. Since node m informs upstream neighbors only about p_j^m, there cannot be two different predecessors at the source i. Therefore, through contradiction, we have proved that having different predecessors is a necessary and sufficient condition for node-disjointedness. □

Using the Theorem 1, instead of checking that each node in an alternate path is not part of any of the implied paths in the successor set, all we need to check is that the predecessor reported is different than the predecessor reported by the rest of the neighbors in the successor set.

The above conditions ensure that each node maintains node disjoint paths from itself to the destination. However, since we only exchange the shortest routes, we do not ensure node disjoint paths across all sources participating in the propagation of a flow from the source to the destination. A simple change in the forwarding mechanism will ensure this. In order to ensure that flows take node-disjoint paths, we require that the source of the flow distribute its packets across all the neighbors in its successor set, but an intermediate hop only uses the best paths for flows it does not originate.

5.5 Route discovery

Route discoveries follow the same procedure as in DST. A node that has no route for a destination starts a route querying cycle. All nodes will receive route queries from all their neighbors thus allowing for multiple routes to the source to be built. On receiving the first query, a node will send out further queries if it does not have a path to the requested destination. If it has a path to the requested destination, then the node sends a reply update which is broadcast to the limited broadcast address, thus allowing all its neighbors to receive it. Therefore unlike in AODV and DSR, the replies are essentially broadcast hop by hop and not unicast back to the source. This

allows for the reply updates to take varied paths en route to the source of the querying. The reply update is differentiated from a regular update by the fact that it has source *pkt.src* set to the requested destination of the route query and a destination *pkt.dst* set to the source of the query.

When node i receives an update, it checks the value of *pkt.dst*. If the value is other than the limited broadcast address, then the update being sent is a reply update, else it is a regular update. The reply update has the same rules as in DST for propagation. A reply update is rebroadcast with the original *pkt.dst* and *pkt.src*, when the following two conditions are met

1. A finite path to *pkt.dst* exists.

2. Distance to *pkt.src* changes from infinite to finite after processing the reply update.

3. The reply update is received from a neighbor whose distance to *pkt.dst* is greater than the node's distance.

These rules help to limit the flooding of the route replies. By forwarding a reply update only when the route to the required destination changes from infinite to finite, the number of updates is reduced at the expense of non-optimal routes. One of the other advantages over other methods of multipath routing [32, 31] is that a node that sends a reply does not need to keep track of who it sends replies to or how many replies it sends.

5.6 Route maintenance

A failure of a link or a link cost change can cause disruption of existing routes. A link coming up does not do that. Since MDST is essentially a on-demand routing protocol, we do not need to use resources for immediate information about new destinations. Regular updates are sent to neighbors when any of the following conditions occurs.

1. Distance to a known destination increases.

2. Next hop to a known destination changes.

3. A node loses the last finite route to a destination.

Distance increases and next-hop changes prompt updates because a loop can occur *only* when a node picks as successor a new neighbor or path that has a higher cost. It is to be noted that updates are sent only when the above changes happen to the shortest path. Changes in all other paths do not prompt updates because these paths were never used or known to upstream nodes.

A source that has a flow and starts a route query cycle now has the advantage of waiting till all its multiple paths are lost before starting a query cycle.

5.7 Packet forwarding

As explained before, a node will only multiplex its flows along its multiple paths if it is the originator of a flow. Otherwise, it will simply send the packets to the next hop in the shortest path.

As shown in the chapter on DST, we use a few more conditions while packet forwarding to prevent permanent loops due to lack of a reliable link layer. In the presence of a reliable link layer, these condition are not needed. A data packet is dropped and a regular update is sent if

A. The data packet is sent by a neighbor that is in the path from the present node to the destination of the data packet.

B. The path implied by the neighbor's distance table entry is different from the path implied in the routing table.

The source of a flow needs to check the first condition for all its paths.

5.8 Effect on TCP of packet reordering in multipath routing

In multipath routing, each of the redundant paths may have a different latency, resulting in packets taking separate paths and arriving out of order, increasing delivery latency and buffering requirements.

Packet reordering causes TCP to believe that loss has taken place when packets with higher sequence numbers arrive before an earlier one. When three or more packets are received before a "late" packet, TCP enters a mode called "fast-retransmit" [9] which consumes extra bandwidth as it attempts to unnecessarily retransmit the delayed packet(s). Hence, reordering can be detrimental to TCP performance.

One way to avoid this behavior is to have intelligence in the forwarding layer to assign all packets of a flow to only one next hop. The other solution is to have

an additional layer between the IP and TCP at the receiver. This layer will reorder packets before sending them up to TCP, thus avoiding the fast retransmit behavior of TCP.

5.9 Performance Evaluation

We implemented MDST and DST in $ns-2$ [17]. To illustrate the effects of link disjointedness, we also implemented a version of MDST termed LMDST, in which the intermediate hops do not multiplex traffic. The MAC layer used is IEEE 802.11 distributed co-ordination function (DCF) for wireless LANs, which uses an RTS/CTS/DATA/ACK pattern for unicast packets and DATA for broadcast packets. The physical layer approximates a 2 Mbps DSSS radio interface. The radio range is 250m. Nodal movement occurs according to the random waypoint model with a speed of 20m/s.

5.9.1 Scenarios used

Our goal with the performance evaluation is two-fold. In the first experiment, we want to compare the performance of MDST, LMDST and DST with respect to end to end delay as node mobilities vary. We also want to see how the control packet overhead changes as multiple paths are taken. In the second experiment, we want to compare the performance of MDST, LMDST and DST as offered load changes. All three protocols use the protocol stack of the wireless node in ns-2.

Both experiments involve 30 node networks in an area of 1000m by 500m and are run for 600 seconds. In the first experiment we run both protocols for various

pause times (0,15,30,60,120,300,600). We have 10 CBR flows that start during the first 100 seconds and last till the end of the simulation. Each flow has a rate of 4 packets/sec with a packet size of 512 bytes. In the second experiment we keep the pause time fixed at 60 seconds. We have 10 CBR flows that start during the first 100 seconds and last till the end of the simulation. We change the flow rate of each flow from 2 packets/sec to 8 packets/sec.

5.9.2 Metrics

We studied the metrics of end-to-end-delay, packet delivery ratio and normalized routing load.

5.9.3 Performance results

Varying mobility

Fig.5.3.a shows the percentage of data packets received for DST, MDST and LMDST as the level of mobility changes. LMDST transports 2% more packets than MDST. This is because each node in LMDST multiplexes traffic resulting in more possible paths. We see that MDST transports 7% more packets than DST because it uses multiple paths. As expected, at pause time 600 all protocols can get 99% of the packets through.

In Fig.5.3.b MDST shows much better delay behavior than LMDST or DST. When packets are distributed along different node-disjoint routes, they face lesser congestion. In addition, in the case of DST packets are waiting in buffers for route querying to complete, resulting in higher delays.

Fig.5.3.c shows the normalized routing load results for varying mobility.

MDST performs better than DST at high mobility chiefly because of lesser queries sent. At pause time 0, MDST has 14% lesser overhead than DST. Having alternate routes helps reduce the number of queries. As the network becomes static (pause time 600) both the protocols have almost the same control overhead because MDST has the same mechanism of querying and replying as DST. The efficiency of MDST and LMDST is almost the same for most mobility scenarios, chiefly because each node has the same overhead for maintaining multiple paths in both the protocols.

Varying load

In this scenario, we keep the pause time static at 60 and number of flows at 10. We change the packet rate of flows from 2 to 8 packets/sec. This test shows us how MDST scales with increase in load.

This is expected to be the case because the mobility of the network is staying constant and most control overhead is in response to links going up and down. The control overhead

Fig.5.5.a shows the percentage of data packet received. As the input packet rate increases MDST delivers more packets than DST simply because it is multiplexing packets along different routes. LMDST transports 2-5% more data packets than MDST at rates higher than 4 packets/sec because each node is multiplexing each flow, resulting in more packets finding their way to the destination.

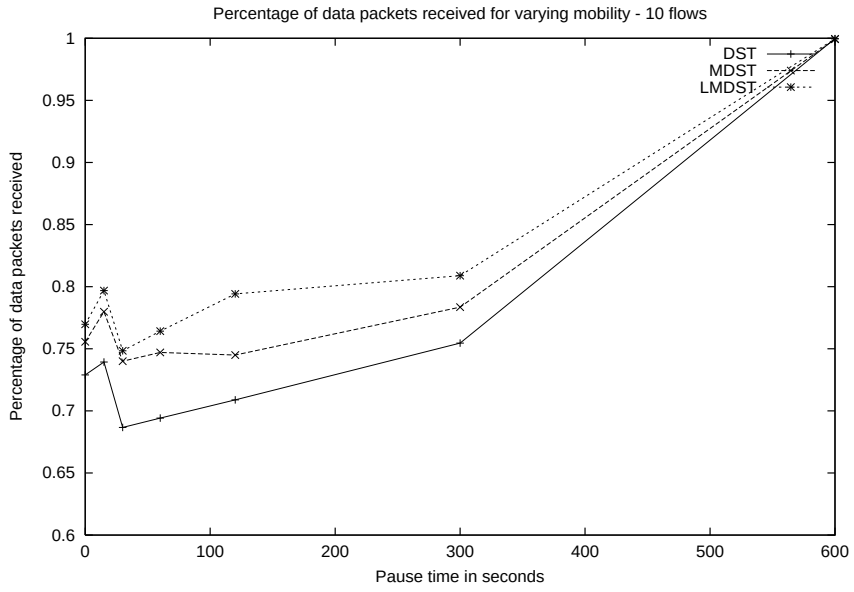
In Fig.5.5.b, we see that at low loads there is very less difference in the delay seen by data packets when MDST, DST or LMDST are run. This is because there is no congestion in the network and the inter-arrival times between the packets is large

enough to not cause backup in queues. However, as the rate of packet arrival into the network goes up, we see that using MDST we incur the least delays.

Fig.5.5.d shows us the normalized routing load of MDST and DST. At low loads the routing load is higher because there are much fewer data packets and that skews the metric. As packet rates increase however, MDST, LMDST and DST show lower normalized routing load. MDST and LMDST perform better than DST at all loads thus showing us the throughput increase achieved by using multipath protocols. The normalized routing load of MDST and LMDST are almost the same, which implies that MDST is an efficient protocols that achieves low delays.

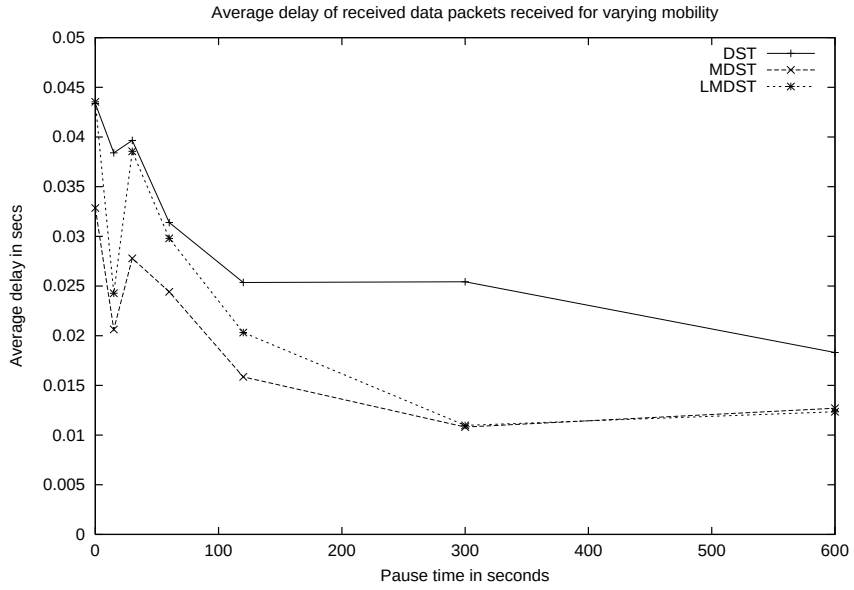
5.10 Conclusions

Multipath routing can be used in ad hoc networks to achieve greater resilience to route failures and better end to end delays. In this chapter we have introduced MDST, an on-demand protocol that extends the source tracing algorithm used in DST to create and maintain multiple node disjoint paths in an ad-hoc wireless network. We use simple rules to keep the multiple paths node disjoint and loop free. Using simulations we studied the performance of MDST relative to DST and LMDST, a link disjoint version of MDST. Our simulation showed that the node disjoint multipath routing introduced in MDST is as efficient as link disjoint and single path routing while achieving better delay performance.



(a)

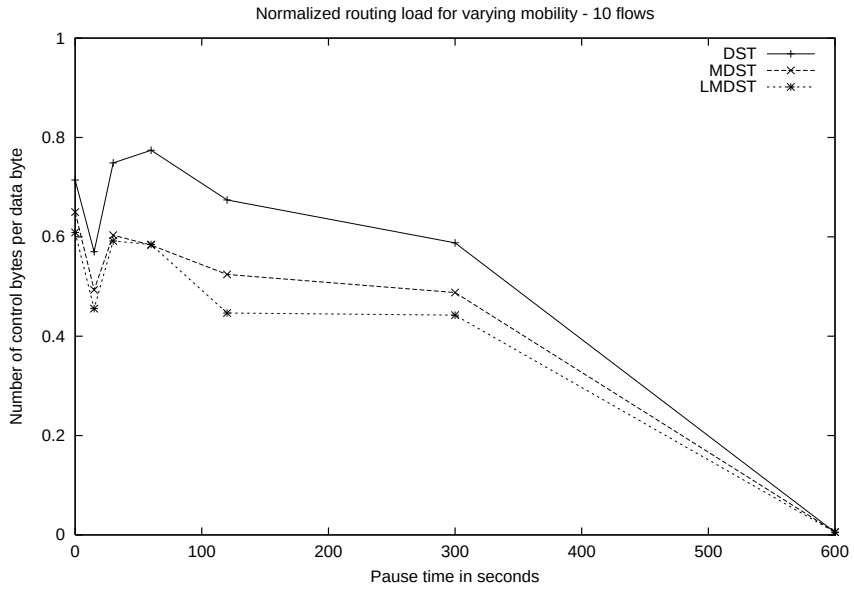
Percentage of data packets received



(b)

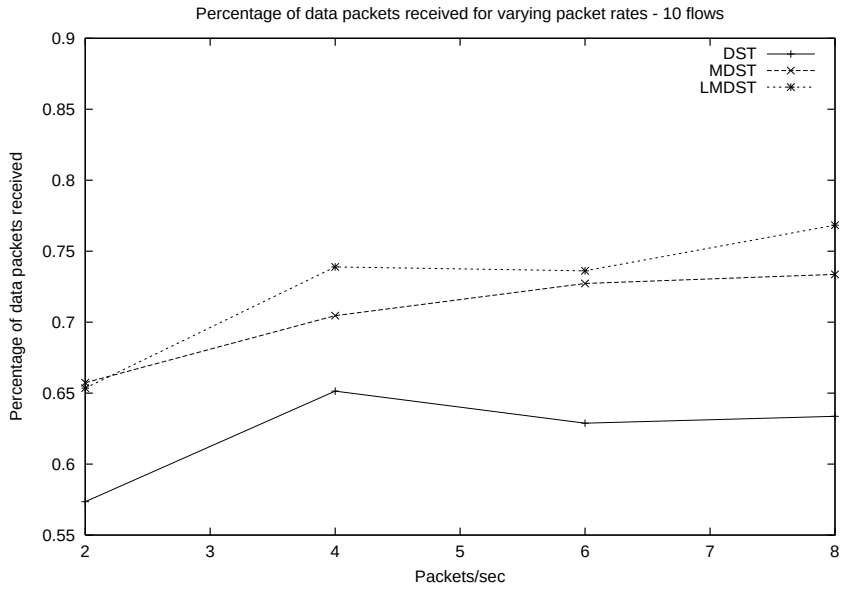
Average end to end delay

Figure 5.3: Results for varying mobility with 10 flows in a 30 node network



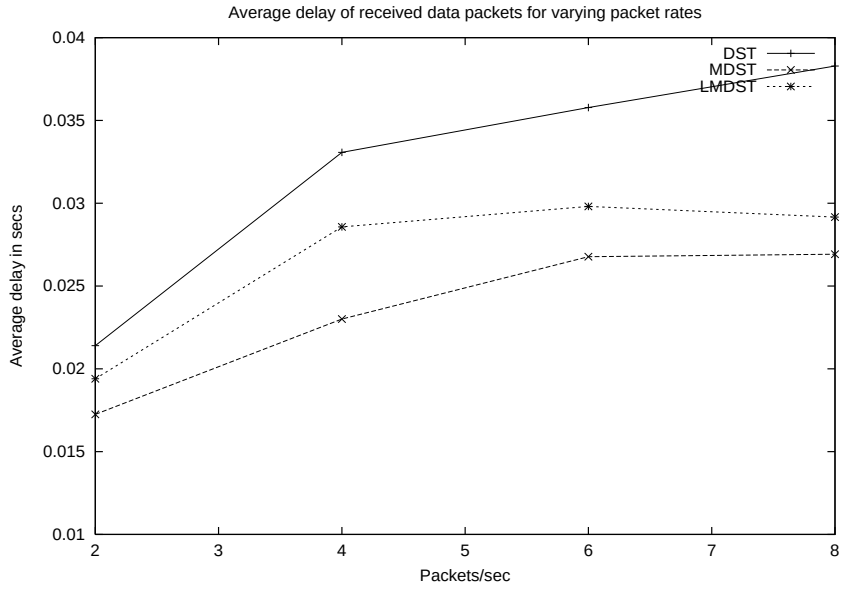
(c)
Normalized Routing Load

Figure 5.4: Results for varying mobility with 10 flows in a 30 node network



(a)

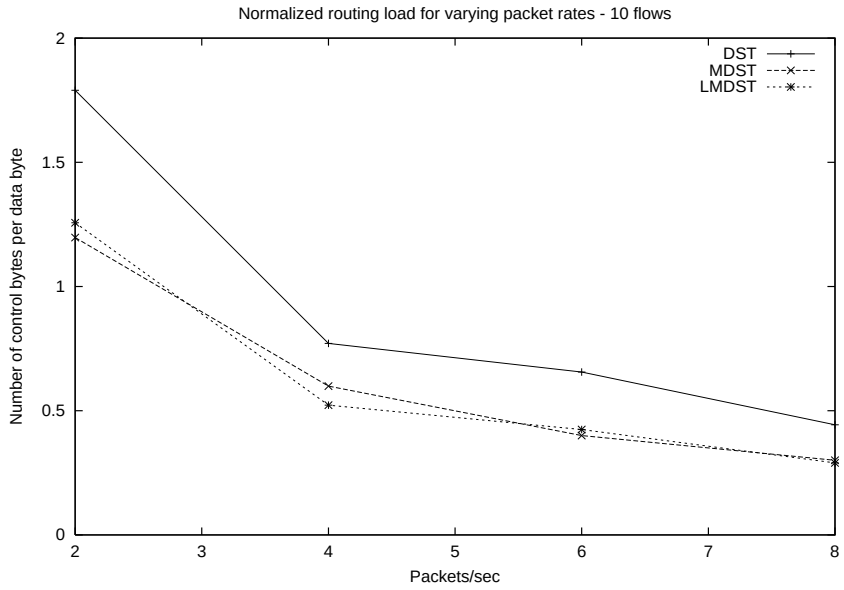
Percentage of data packets received



(b)

Average end to end delay

Figure 5.5: Results for varying load with 10 flows in a 30 node network



(c)
Normalized Routing Load

Figure 5.6: Results for varying load with 10 flows in a 30 node network

Chapter 6

Summary and Future Work

6.1 Contributions

The goal of this thesis has been to design robust, efficient and scalable unicast routing solutions for ad-hoc networks. We explored protocols that used distance vector techniques and made considerable progress in understanding how they could be used to incur low overhead while providing high packet delivery rates.

We have presented ROAM, the first on-demand routing algorithm that provides multiple loop-free paths without the need for complete path information, sequence numbers refreshed periodically, or time stamps. We have proved that ROAM is loop-free and converges in a finite time. We addressed the "searching to infinity" problem and eliminated its occurrence in ROAM, such that sources do not send repeated flood searches in the event of destinations being unreachable. We presented the time and communication complexity results for ROAM. ROAM is very applicable on wired networks, wireless networks with static nodes, and could also be applied to wireless networks with some degree of mobility.

In the second part of this thesis we introduced and analyzed the DST pro-

tocol, which constitutes a new approach for on-demand distance vector routing for ad-hoc networks with unreliable delivery. DST uses a source-tracing algorithm similar to the one advocated in prior table-driven routing protocols in which routers maintain routing information for all network destinations. To reduce the number of loops, the source-tracing algorithm allows for complete paths to be checked for loops before being added to the routing table. Since DST does not use sequence numbers, it is not prone to inefficiencies in the presence of node failures. DST also does not use reliable updates or polling of neighbors. This implies that DST creates substantially less overhead than protocols that use the above features. We introduced conditions that reduce control packet overhead at the expense of non-optimal routes, all the while preventing permanent looping of data packets. We presented a proof of correctness which showed that in the presence of reliable updates, the non-optimal route conditions would create loop-free routes and the protocol would converge in finite time. Simulations in CPT were used to compare DST with DSR, which is one of the most efficient on-demand routing protocols reported in literature. In most high mobility scenarios, DST showed an order of magnitude less control overhead than DSR. For other scenarios, it performed consistently better than DSR with respect to control overhead. The results for delay, hop count and percentage of data packets received are mixed. In some cases DSR performed better and in others DST performed better, which led us to believe that both protocols are at par when performance in these metrics is taken into consideration. Our simulations results showed that DST is very suitable for ad-hoc networks and incurs limited control overhead, even in cases of high mobility.

We also implemented DST in ns2 in order to compare DST with DSR and AODV. The ns-2 simulations showed that DST is very efficient protocol for high flow scenarios regardless of mobility. AODV is a more efficient protocol in high mobility combined with low load scenarios. Regardless of the number of flows and mobility, DST had the best delay behavior. This leads us to suggest that DST is an ideal protocol for delay sensitive applications like audio flows.

In the third part of the thesis, we aimed is to study the effect of relaxing the requirement for shortest paths in a table driven routing protocol and determining if this can lead to solutions whose performance is equivalent to on-demand routing approaches. Our goal was to design a table-driven distance-vector routing protocol that uses the same constraints used in on-demand routing protocols, i.e. paths are used as long as they are valid and updates are only sent when a path becomes invalid. We introduced BEST, which is bandwidth efficient table driven routing protocol that uses source tracing. Simple rules were introduced to prevent long term looping in the absence of reliable delivery of packets. The rules used in BEST are simpler than those introduced for STAR [22], which is the only other table-driven routing protocol that has been shown to be as efficient as on-demand routing protocols. We have presented a proof of correctness for BEST under the assumption of reliable delivery of packets. Simulations in CPT were used to compare DST, BEST and DSR. Simulations on various scenarios suggested that BEST is a good alternative for community networks and networks than are running QoS based applications, where delay is of chief concern.

The last part of the thesis introduced MDST, which is an on-demand algo-

rithm that uses source tracing to create and maintain multiple paths in an ad-hoc wireless network. We used simple rules to keep the multiple paths node disjoint and loop free. We showed that with lesser control packet overhead than DST, we maintain multiple paths that help reduce delay and increase throughput seen by data packets. We also showed than node disjoint paths perform better than link disjoint paths in terms of delay seen by data packets.

6.2 Future Work

Our work in multipath is a first step towards designing a QoS framework for ad-hoc networks. As shown in the work by Cidon et al. [6] the reservation establishment time using multiple paths is significantly lower than the reservation establishment times using single paths. Ultimately, QoS in ad-hoc networks will be intimately connected with the kind of scheduling available at the medium access layer. The combination of resource reservation along paths with the dynamic schedules at the individual nodes will help provide delay guarantees that are essential for any multimedia application. The routing protocols themselves need to be adapted to use fast changing metrics like delay and available bandwidth.

As ad hoc networks become larger, it will be impossible for any type of protocol, table-driven or on-demand to scale efficiently. At this point, it will become essential to reduce the amount of routing information stored and exchanged by introducing hierarchies. Furthermore, if these hierarchies are self-addressable like the Landmark hierarchies [50], it will help mitigate the address assignment problem.

Another important issue that is gaining prominence in ad-hoc networks is security. In simple ad-hoc network all entities are trusted. Eventually, the design on routing protocols will have to incorporate the concept of trusted nodes and key management. The routing protocol will also have to be evaluated to ensure that they do not inadvertently create situations that may help intruders. One such simple issue we solved with ROAM was searching to infinity which could be used as the basis of a denial of service attack.

Routing in ad hoc networks can be further specialized to do anycasting. This will be useful for large ad hoc networks where certain specialized nodes provide services like DNS and Internet access and one needs to locate only the closest such node. Such network architectures also create the need for hybrid routing solutions where the network proactively maintains routes to important nodes and reactively maintains routes to nodes that offer no special services.

Bibliography

[1] L. Bao and J.J. Garcia-Luna-Aceves. A New Approach to Channel Access Scheduling for Ad Hoc Networks. In *Proceedings of ACM Mobicom'01*, Rome, Italy, July 2001.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall Inc., 2nd edition, 1992.

[3] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. The Institute of Electrical and Electronics Engineers, 1997. IEEE Std 802.11.

[4] A. Nasipuri et al. On-Demand Multipath Routing for Mobile Ad-Hoc Networks. In *Proceedings of IEEE ICCCN'99*, Boston, MA, October 99.

[5] D. Sidhu et al. Finding Disjoint Paths in Networks. In *Proceedings of ACM Sigcomm'91*, pages 43–51, Zurich, Switzerland, September 1991.

[6] I. Cidon et al. Analysis of Multipath Routing. *IEEE/ACM Transaction on Networking*, 7(6):885–896, December 1999.

[7] J. Broch et. al. A Performance Comparison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols. In *Proceedings of ACM Mobicom'98*, Dallas, TX, October 1998.

[8] J. Chen et al. An Efficient Multipath Forwarding Method. In *Proceedings of IEEE Infocom'98*, pages 1418–1425, San Francisco, CA, March 1998.

[9] M. Allman et al. TCP Congestion Control. Technical Report RFC 2581, April 1999.

[10] M.R. Pearlman et al. On the Impact of Alternate Path Routing for Load Balancing in Mobile Ad Hoc Networks. In *Proceedings of ACM Mobihoc'00*, pages 3–10, 2000.

[11] N. Taft-Plotkin et al. Quality-of-Service Routing using Maximally Disjoint Paths. In *Proceedings of IEEE IWQoS'99*, pages 119–128, London, UK, June 1999.

[12] P. Johansson et al. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks. In *Proceedings of IEEE/ACM Mobicom '99*, pages 195–206, Seattle, WA, Aug. 1999.

[13] R. Dube et al. Signal Stability-Based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks. *IEEE Personal Communication*, February 1997.

[14] R. E. Kahn et al. Advances in Packet Radio Technology. *Proceedings of the IEEE*, 66(11):1468–1496, Nov 1978.

[15] R. Ogier et al. Distributed Algorithms for computing Shortest Pairs of Disjoint Paths. *IEEE Transactions on Information Theory*, 39(2):443–455, March 1993.

[16] S. R. Das et al. Comparative Performance Evaluation of Routing Protocols for Mobile Ad-Hoc Networks. In *Proceedings of ICCCN'98*, pages 153–161, Lafayette, LA, Oct. 1998.

[17] K. Fall and K. Varadhan. *Ns notes and documentation.* The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC, 2002. Available from http://www.isi.edu/nsnam/ns/.

[18] C.L. Fullmer and J.J. Garcia-Luna-Aceves. Solutions to Hidden Terminal Problems in Wireless Net works. In *Proceedings of ACM Sigcomm'97*, Cannes, France, September 1997.

[19] E. M. Gafni and D.P. Bertsekas. Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology. *IEEE Transactions on Communications*, COM-29(1):11–18, January 1981.

[20] J. J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Transactions on Networking*, 1(1):130–141, Feb 1993.

[21] J.J. Garcia-Luna-Aceves and J. Raju. Distributed Assignment of Codes for Multihop Packet Radio Networks. In *Proceedings of IEEE Milcom'97*, Monterey, CA, 1997.

[22] J.J. Garcia-Luna-Aceves and M. Spohn. Source-Tree Routing in Wireless Networks. In *Proceedings of IEEE ICNP'99*, Toronto, Canada, 1999.

[23] J.J. Garcia-Luna-Aceves and A. Tzamaloukas. Reversing The Collision-Avoidance Handshake in Wirele ss Networks. In *Proceedings of ACM/IEEE Mobicom'99*, Seattle, Washington, August 1999.

[24] Z. Haas and M. Pearlman. The Zone Routing Protocol for Highly Reconfigurable Ad-Hoc Networks. In *Proceedings of ACM Sigcomm'98*, Vancouver, British Columbia, August 1998.

[25] C. Huitema. *Routing in the Internet*, chapter 6, pages 145–148. Prentice Hall PTR, 1995.

[26] J. M. Jaffe and F. H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Transactions on Communication*, COM-30(7):1758–1762, July 1982.

[27] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad-Hoc Wireless Networks. *Mobile Computing*, 1994.

[28] L. R. Floyd Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, 1962.

[29] J. Ju and V.O.K. Li. An Optimal Topology-Transparent Method in Multihop Pa cket Radio Networks. *IEEE/ACM Transactions on Networking*, 6(3), June 1998.

[30] J. Jubin and J. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, 1987.

[31] S.J. Lee and M. Gerla. Split Multipath Routing with Maximally Disjoint Paths in Ad-Hoc Networks. In *Proceedings of IEEE ICC'01*, pages 3201–3205, 2001.

[32] M. Marina and S. Das. On-demand Multipath Distance Vector Routing in Ad Hoc Networks. In *Proceedings of IEEE ICNP'01*, pages 14–23, 2001.

[33] N.F. Maxemchuk. Dispersity Routing. In *Proceedings of IEEE ICC'75*, pages 41:10–41:13, 1975.

[34] J. Moy. *OSPF Version 2*, 1991. RFC 1247.

[35] S. Murthy and J. J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *ACM Mobile Networks and Applications Journal*, pages 183–197, Oct. 1996.

[36] S. Murthy and J.J. Garcia-Luna-Aceves. A More Efficient Path Finding Algorithm. In *Proceedings of 28th Asilomar Conference*, Pacific Grove, CA, October 1994.

[37] S. Murthy and J.J. Garcia-Luna-Aceves. Congestion-Oriented Shortest Multipath Routing. In *Proceedings of IEEE Infocom'96*, pages 1028–1036, San Francisco, CA, March 1996.

[38] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of the IEEE Infocom'97*, Kobe, Japan, April 1997.

[39] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *Computer Communication Review*, pages 234–244, Oct 1994.

[40] C. E. Perkins and E. M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications.*, pages 90–100, Feb 1999.

[41] J. Raju and J.J. Garcia-Luna-Aceves. A new approach to on-demand loop-free multipath routing. In *Proceedings of ICCCN'99*, Natick, MA, 1999.

[42] J. Raju and J.J. Garcia-Luna-Aceves. A Comparison of On-Demand and Table-Driven Routing for Ad-Hoc Wireless Networks. In *Proceedings of ICC'00*, New Orleans, Louisiana, June 2000.

[43] J. Raju and J.J. Garcia-Luna-Aceves. Efficient On-Demand Routing Using Source-Tracing in Wireless Networks. In *Proceedings of Globecom'2000*, San Francisco, CA, 2000.

[44] J. Raju and J.J. Garcia-Luna-Aceves. Scenario based comparison of Source Tracing and Dynamic Source Tracing Protocols for Ad-Hoc Networks. In *Proceedings of ICC'01*, Helsinki, Finland, 2001.

[45] J. Raju and J.J. Garcia-Luna-Aceves. Scenario based comparison of Source Tracing and Dynamic Source Tracing Protocols for Ad-Hoc Networks. *ACM Computer Communication Review - Special Issue on wireless extensions to the Internet*, October 2001.

[46] N. Shacham and J. Westcott. Future Directions in Packet Radio Architectures and Protocols. *Proceedings of the IEEE*, 75(1):83–98, Jan 1987.

[47] T. Shepard. A Channel Access Scheme for Large Dense Packet Radio Networks. In *Proceedings of ACM Sigcomm'96*, Aug 1996.

[48] Z. Tang and J.J. Garcia-Luna-Aceves. Hop-Reservation Multiple Access (HRMA) for Ad-Hoc Net works. In *Proceedings of IEEE Infocom'99*, March 1999.

[49] C-K. Toh. A Novel Distributed Routing Protocol to support Ad-Hoc Mobile Computing. In *Proceedings of the IEEE 15th Annual Int'l Phoenix Conference Computers and Communications*, pages 480–86, Mar. 1996.

[50] P.F. Tsuchiya. Landmark routing algorithms: Analysis and Simulation Results. Technical Report Technical Report MTR-89W00277, MITRE Corporation, December 1989.

[51] S. Vutukury and J.J. Garcia-Luna-Aceves. An algorithm for Multipath Computation using Distance Vectors with Predecessor Information. In *Proceedings of IEEE ICCCN'99*, pages 534–539, Boston, MA, October 1999.

[52] W.T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-free multipath routing using generalized diffusing computations. In *Proceedings of IEEE Infocom 98*, pages 1408–1417, San Francisco, CA, March 1998.

[53] C. Zhu and S. Corson. A Five-Phase Reservation Protocol (FPRP) for Mobile A d Hoc Networks. In *Proceedings of IEEE Infocom'98*, 1998.